MacTech Magazine™

Formerly MacTutor

FOR MACINTOSH PROGRAMMERS & DEVELOPERS

DECEMBER 1994 • VOLUME 10, No. 12

SPECIAL 10TH ANNIVERSARY ISSUE!

In This Issue!

ETTING STARTED
he Required Apple Events

MPROVING THE FRAMEWORK
Better Window Management
n MacApp

THINK TOP 10

PROGRAMMER CHALLENGE:
Rubik's Cube

ENHANCING SPROCKET
Adding Threads To Sprocket

NEW APPLE TECHNOLOGY
Writing Control Strip Modules

TOOLS OF THE TRADE
Object Master 2.5 Universal

TOOLS OF THE TRADE
QC: Test for Success

THE VERY FIRST ISSUE!
Ten Years Doesn't Seem Like So Long Ago

UNIVERSAL RESOURCE LOCATORS

AND MORE!

$5.85 US
$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

12

0 32128 74887 8

NEW SYMANTEC C++ 7.0. YOU'D HAVE TO BE BLIND TO PROGRAM MACINTOSH WITH ANYTHING ELSE.

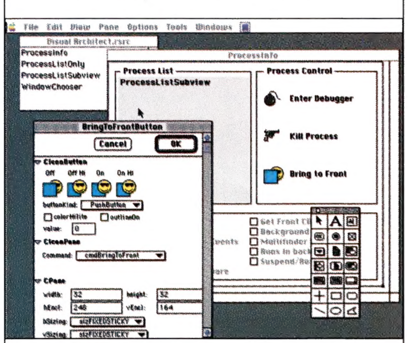
We've got something to show you that you will have to see with your own eyes to believe.

It's Symantec C++ 7.0. And if you're a Macintosh developer, you'll want to take a good look.

VISUAL ARCHITECT. THE EASIEST WAY TO DESIGN A MACINTOSH INTERFACE.

Our new Visual Architect lets you visually design and create all of your windows, dialogs, bitmaps, alerts, controls and menus. Then it automatically generates your complete Mac application.

You can test the user interaction of your Macintosh application immediately while in Visual



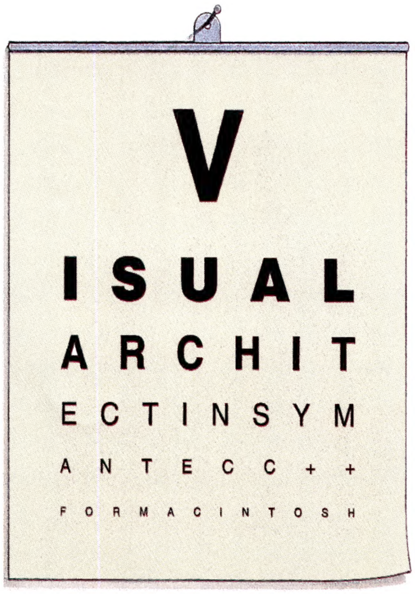
Visual Architect simplifies the creation of your Macintosh interface so you can focus on the big picture: creating killer applications.

Architect's prototype mode, without even having to compile your application.

Completely integrated within our THINK Project Manager environment, Visual Architect is immediately accessible to you every step of the way. Simplifying your entire development process from design to the final code.

To help cut your development cycle even further, we've also included a brand new object browser that more efficiently lists, examines and modifies all objects instantiated during the course of your applications development.

We've also enhanced our THINK Class Library 2.0 (TCL 2.0) with C++ pointers and memory management, new support for AppleEvents and scriptable apps, exception handling support and of course, object-persistence.



Any way you look at it, Symantec C++ 7.0 for Macintosh with its Visual Architect is the easiest way to design the Macintosh user interface.

THE PATH TO POWER MACINTOSH.

Our new TCL 2.0 uses universal headers, removes all the assembly code specific to 68K and it also supports Power Macintosh procedural pointers.

So all of your TCL applications written in Symantec C++ 7.0 will port easily to Power Macintosh giving you a head start on your Power Macintosh development.

What's more, we're offering you a developers' pre-production release of our Power Macintosh Cross Development Kit. It has the tools you need to get started on creating native Power Macintosh applications today, including a copy of Apple's Power Macintosh linker and debugger.

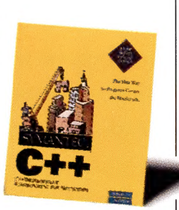
It's a way to speed up the development of your Macintosh applications right now – and be the first to move your programs to the Power Macintosh platform.

CALL 1-800-628-4777.

Ask for ext.9H18 to upgrade to Symantec C++ 7.0 for Macintosh today for just $149.95* and get our developers' pre-production release of the Power Macintosh Cross Development Kit for just $100.

This offer is not available in any retail store. So call now.

And get your code ready for Power Macintosh.



SYMANTEC.®

*$499.00 Suggested Retail Price. Power Macintosh Development Kit available only with purchase of or upgrade to Symantec C++ 7.0. For more information in Canada, call 1-800-667-8661. Symantec, the Symantec logo, Symantec C++, and Visual Architect are trademarks of Symantec Corporation. All other products or brand names are trademarks of their respective holders.

Time Is On Your Side

Now VIP-C creates applications that are accelerated for Power Macintosh.

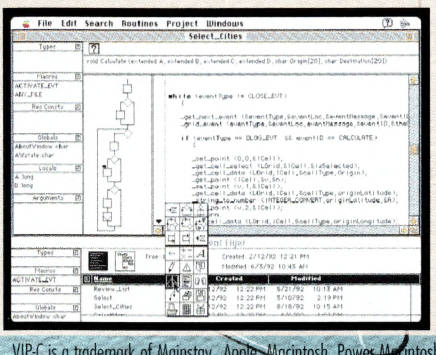
Now, create 68K and Power Macintosh applications faster than ever before. VIP-C's visual design, its application building aids, and its ability to let you modify your program code and interface with immediate feedback, give you a real programming edge. Read further to see why, with VIP-C, time really *is* on your side...

Complete Development System

Now develop with just one comprehensive system. VIP-C integrates all the tools you need to create stand-alone 68K and Power Macintosh applications—right out of the box! Develop applications in a standard language using this intelligent, full-featured Rapid Application Development (RAD) environment. No more moving from tool to tool. No more dependence on non-standard languages.

For All Levels of Programmers

VIP-C offers multiple levels of support to best fit your abilities. If you just want to type C code into its editor, you can—and VIP-C will automatically check the syntax and create a flowchart of your code. For higher-level support, VIP-C features prewritten intelligent prototypes of all the Mac Toolbox calls, high-level VIP-C Functions that simplify low-level Toolbox calls, Resource Editors that let you visually design and create your windows, menus, dialogs, buttons, etc., and even a VIP-C Dispatcher to provide an efficient and reliable application framework—all in one complete development environment.



```
void Calculate (extended A, extended B, extended C, extended D, char Origin[20], char Destination[20])

while (eventType != CLOSE_EVT)
    {

    _get_next_event (&eventType,&eventLoc,&eventMessage,&eventI
    _grid_event (eventType,&eventLoc,eventMessage,&eventID,&the

    if (eventType == DLOG_EVT  && eventID == CALCULATE)
        {

        _set_point (0,0,&lCell);
        _get_cell_select (LGrid,&lCell,&isSelected);
        _get_cell_data (LGrid,lCell,&cellType,Origin);
        _get_point (lCell,&v,&h);
        _set_point (v,1,&lCell);
        _get_cell_data (LGrid,lCell,&cellType,originLatitude);
        _string_to_number (INTEGER_CONVERT,originLatitude,&A);
        _set_point (v,2,&lCell);

        _get_cell_data (LGrid,lCell,&cellType,originLongitude);
```

User Interface Generation Tools

The VIP-C Dispatcher simplifies program development by automating the main event loop. It acts as a central controller to manage program events by distributing tasks to different routines. User interface items—menus, dialogs, buttons, etc.—are created with integrated resource editors that automatically link user action to your code.

VIP-C 1.5
Everything you need to create complete Macintosh and Power Macintosh applications in one box.

Powerful Prewritten Functions

Functions for the most frequently used features of the Mac interface have been prewritten and stored in a palette for instant use. With the click of a mouse you can lighten your programming workload and speed design and coding time by several orders of magitude.

On-Line Macintosh Toolbox Calls

VIP-C accesses the complete Mac Toolbox. Search and display prototypes for any function, structure, or macro. No more rooting through multiple volumes of Inside Macintosh!

Attention Database Developers!

VIP-C users now have the power to create commercial quality, multi-user, relational databases using the new VIP Database Manager (sold separately). VIP Database Manager has the built-in functionality of the C-Index Pro database engine (recently chosen by Apple OSA for creating the eWorld content publishing software) giving you an inexpensive way to create professional databases, royalty-free!

Compiler Hotlink

If you have a favorite complier, use it! Compile and run your application directly from a menu item in the VIP-C editor. VIP-C supports: THINK C, MPW-C, and CodeWarrior (each sold separately).

Buy VIP-C now for a Limited-Time Special Price of only $295. Act now and get VIP Database Manager for only $149.

VIP-C's suggested retail price is $495. But from now until Dec. 31, 1994, you can order VIP-C for $295. Call Mainstay to order today.

Mainstay

(805) 484-9400

591-A Constitution Ave.
Camarillo, CA 93012
fax: (805) 484-9428

71 rue des Atrébates
B-1040 Brussels, Belgium
32-2/733.97.91

MacTech MAGAZINE Formerly MacTutor
FOR MACINTOSH PROGRAMMERS & DEVELOPERS
A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?** If you have any questions, feel free to call us at 310/575-4343 or fax us at 310/575-0925.

DEPARTMENTS	Internet	CompuServe	AppleLink	America Online	GEnie
Orders, Circulation, & Customer Service	custservice@xplain.com	71333,1063	MT.CUSTSVC	MT CUSTSVC	MACTECHMAG
Editorial	editorial@xplain.com	71333,1065	MT.EDITORIAL	MT EDITORS	–
Programmer's Challenge	progchallenge@xplain.com	71552,174	MT.PROGCHAL	MT PRGCHAL	–
Ad Sales	adsales@xplain.com	71552,172	MT.ADSALES	MT ADSALES	–
Accounting	accounting@xplain.com	–	–	–	–
Marketing	marketing@xplain.com	–	–	–	–
Press Releases	pressreleases@xplain.com	–	–	–	–
General	info@xplain.com	71333,1064	MACTECHMAG	MacTechMag	–
Online support area	ftp://ftp.netcom.com/pub/xplain	*type* GO MACTECHMAG	*see* Third Parties: Third Parties (H-O)	*use keyword:* MACTECHMAG	–

MACTECH MAGAZINE

Publisher Emeritus • David Williams
Editor-in-Chief/Publisher • Neil Ticktin
Editor • Scott T Boyd
Assistant Editor • Mary Elaine Califf
Technical Editor • Ken Gladstone
Advertising Executive • Ruth Subrin
Art Direction • Judith Chaplin, Chaplin & Assoc.

XPLAIN CORPORATION

VP Finance & Operations • Andrea J. Sniderman
Customer Service • Al Estrada
Software Engineer • Don Bresee
Accounting Assistant • Brian Shin
Administrative Assistant • Susan Pomrantz
Board of Advisors • Blake Park, Alan Carsrud

AUTHORS & REGULAR CONTRIBUTORS

MacTech Magazine is grateful to the following individuals who contribute on a regular basis. We encourage others to share the technology. We are dedicated to the distribution of useful programming information without regard to Apple's developer status. For information on submitting articles, ask us for our **writer's kit** which includes the terms and conditions upon which we publish articles.

Richard Clark
General Magic
Mountain View, CA

Chris Espinosa
Apple Computer, Inc.
Cupertino, CA

Jörg Langowski
Jörg's Folder
Grenoble Cedex, France

David R. Mark
M/MAC
Arlington, VA

Jordan Mattson
Apple Computer, Inc.
Cupertino, CA

Mike Scanlin
Programmer's Challenge
Mountain View, CA

Symantec Technical Support Group
THINK Division
Eugene, OR

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 1617 Pontius Avenue, 2nd Floor, Los Angeles, CA 90025-9555. Voice: 310/575-4343, FAX: 310/575-0925. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Second Class postage is paid at Los Angeles, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 250055, Los Angeles, CA 90025-9555.

TABLE OF CONTENTS

December 1994 • Volume 10, No. 12

MacTech MAGAZINE™ Formerly MacTutor
FOR MACINTOSH PROGRAMMERS & DEVELOPERS

THE EDITOR'S PAGE

By Scott T Boyd, Editor

Getting On The Net

I work out of my home here on the Pacific coast. With five phone lines, five computers on an Ethernet, three modems, two cats, an eighteen-month-old daughter, a wife, and views of the ocean and a mountain, we may be looking at something like the home office of the future. No home office could be complete, though, without getting wired, and that takes more than caffeine.

Since January, I've experimented with a variety of schemes for connecting. I have all sorts of accounts on all the online services, but none of them qualify as "wired". Getting connected, in this case, means plugging in to the Internet.

I started off by doing what all the Apple escapees seemed to be doing – I got a Netcom PNC (Personal Network Connection). I could use MacTCP and run all of my favorite Macintosh apps. It also came with a unix shell account and an e-mail address. It cost $17/month plus an hourly fee for every hour past some allotment.

Flocking to a net provider along with a few bazillion other subscribers isn't always the best plan. They were growing so fast that they couldn't add equipment, staff, and phone lines fast enough. I found a good excuse to find a new provider – we moved north to Montara (between Half Moon Bay and San Francisco).

Once here, I hooked up with QuakeNet. They're small, know their stuff, and answer the phone when something goes wrong (hey, it's networking; something's bound to go wrong from time to time). Now I pay $85/month for a 14.4Kbps full-time connection. It's a local call, so I can keep my modem connected constantly for the cost of a residential phone line. I find myself talking on the phone less, and I expect rates to drop as competition heats up. It's great to have an unmetered connection – I never have to think about what it's costing me to stay connected.

I've teamed up with a few other home office workers, and we're finding all sorts of ways to use the net. We've tunneled AppleTalk to share printers and servers. We've set up FTP and chat servers so we can talk to each other and exchange files. We've also experimented with sending voice over the net. With a little work, we'll be able to talk over 14.4 connections, and that will also reduce our phone bills.

We got our start using MacPPP (similar to SLIP) to make a point-to-point serial connection to our providers. It's no more complex to configure than MacTCP – straightforward, but not trivial. A couple of us are now using MachTen (Tenon's unix product for Macs) to make the PPP connection and do ip routing. That's more fun because now I can use any of the machines on my net, as well as offer dialup service to friends in the area. It's also a lot more work because it involves configuring and administering a unix box (e.g. sendmail, FTP, DNS, accounts, firewalls, etc...). You haven't seen bad documentation until you've tried to run a unix system. Flexibility breeds complexity. Turnkey it's not.

I'd like to hear from those of you who have gotten on the net and found fun ways to use it. I'd especially like to hear from those of you who have written Macintosh-based tcp/ip software to take advantage of the net. The Net offers more opportunity than just running Mosaic.

Software Piracy

Have you ever wondered about those numbers we see from the Software Publishers Association and the Business Software Alliance about software piracy? BSA claims that global piracy costs over $12 billion dollars each year. *Where do they get these numbers?* They throw them around as if they had a basis for their claims. If so, why don't they document how they arrive at those numbers? Might it be that the numbers wouldn't impress us so much if we knew how they were derived?

Piracy certainly exists, but $12 billion? They do our industry a disservice by making unsubstantiated and unbelievable claims. That doesn't do any of us any good in the long run. Many of you work for member companies; your opinion might help these groups wise up.

Do You Do Windows?

We all hear that Windows is going like gangbusters. It's hard to tell whether it's hype or excitement, but there sure seems to be a lot of interest in Windows95 (formerly known as the late Chicago). Have you felt the urge or the need to do Windows?

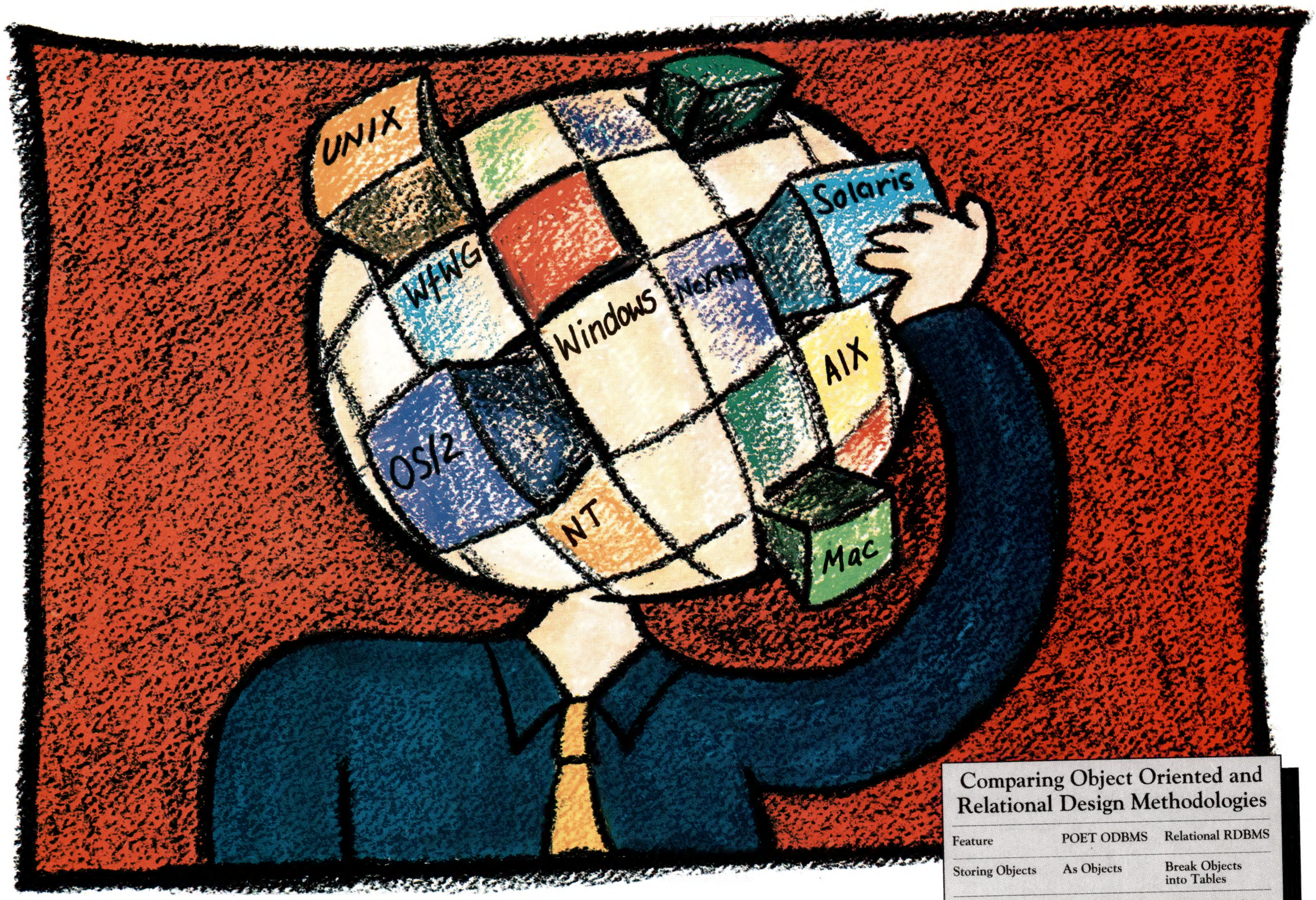
I know developers who have made the switch so completely that they don't do Macintosh any more. They couldn't see missing out on the opportunity to sell their products into the much larger market that Windows seems to offer. They aren't doing Macintosh mostly because they're small and can't afford to split their efforts.

Other developers have decided to add Macintosh to the list of platforms they support. How else can you increase your revenue once you have established how much Windows marketshare you can grab? These vendors see the addition of a platform as a simple money decision. Hire one or two Mac programmers and have them port the Windows code. No big deal (but no love for the platform, either).

Others have successfully built a product and a market on the Macintosh. Having worked hard to own as much market

Continued on page 82

Don't even think about using a Relational Database System !



Comparing Object Oriented and Relational Design Methodologies

Feature	POET ODBMS	Relational RDBMS
Storing Objects	As Objects	Break Objects into Tables
Database Model	User Application Model	Separate Database Model Required
C++ Integration	Total	Poor
Database Operations	At Object Level	Must Write Code
Productivity	Increased	Reduced
Complex Object Performance	Excellent	Poor

Use the POET Object Database for C++

C++ and class libraries have made the GUI development much easier. After all, it is only logical that more and more developers think in objects. But object orientation shouldn't end at the user interface programming level.

The Problem: Without POET, a C++ programmer must use flat files or a RDBMS to store objects. He has to write code to overcome the mismatch between the application and the database model. This leads to design restrictions, performance penalties and more code to write and maintain.

The Solution: POET operates at the object level, it speeds up the development process and provides greater performance. Furthermore, the developer can simply take the object-oriented application design in C++ and map it 1 to 1 into the database. Without any compromise at all!

Full featured database: POET provides complete support for Encapsulation, Inheritance, Polymorphism and Containers as well as queries, object locking, and transaction handling .

True cross platform support: Complete interoperability makes the development of network enabled applications a breeze. POET supports Macintosh, Power Macintosh, Windows 3.1, Windows NT, Win32s, Windows for Workgroups, Novell Netware (NLM), SUN, AIX, HP-UX, SGI, SCO, OS/2 and NeXTStep.

Call 1-800-950-8845

POET
Software

POET Software Corporation, 4633 Old Ironsides Dr., Suite 110, Santa Clara, CA 95054, Tel.: (408) 970-4640 Fax: (408) 970-4630
United Kingdom (Silicon River) +44 81-316-7777, Germany (POET Software) +49 40-60-99-00, Australia (Microway) +61-3-580-1333, France (LCI) +33-1-34-65-7777
Netherlands (Protools Software) +31 20 645 5023, Sweden (Duke Systems) +46 8 703 2781

THE PUBLISHER'S PAGE

By Neil Ticktin, Publisher/Editor-In-Chief

STARTING THAT SECOND DECADE...

As I'm sitting here writing this column, I'm reflecting on our industry over the past decade. Maybe it's the comparison of how we work today vs. 1984. Maybe it's the fast pace of change in today's industry. Maybe it's the opportunities we have before us. Or, maybe...just maybe, I just haven't had enough sleep.

Right now, I'm sitting at a vacation resort with my Duo 280c with 18 megabytes of RAM, 300 meg of disk, a CD drive, and a DeskWriter – logged into ARA at 14.4 and handling e-mails at a furious rate (175 today alone), copying files in the background and writing this page in QuarkXPress®. In a matter of hours, this page will be ready to send electronically to the printer to make film as we bring this issue to life. What a far cry from the days when MacTech/MacTutor was first published with a 128k Mac, no hard drive, articles written in MacWrite, printed on an ImageWriter, pasted up with a glue gun, and then photographed with a conventional pre-press camera. Our industry – the computer industry – is *truly* impressive. Where else could so much change so quickly?

As many of you know, ten years ago, the original MacTech was founded by David and Laura Smith. Their goal was to disseminate programming information about the Macintosh without regard to Apple's Developer status. The original editorial board consisted of folks who shared a similar vision – they wanted to help the Macintosh succeed. Today, long after they sold the magazine, the Macintosh developer community is not much different – we still love the machine, and MacTech still doesn't care about your "developer status."

As we start our second decade at MacTech Magazine, we are riding an unprecedented era of growth, strength, and improvement. Most of you are familiar with the editorial changes that we started 18 months ago and accelerated this year. Today, our staff is nearly three times what it was in 1992, and we just hired another person for our editorial staff. And while a typical magazine loses 40-60% of its readership each year (yes, it really is that much), our subscriber base has grown 45% net in the last 18 months.

But, while growth is great, it's not everything. Today, MacTech is providing you more of what you wanted. While editorial quality is clearly a critical part of what we do for you, there's more. In the last few years, we've implemented a number of things for you because you asked for them. These ranged from online support to adding the date to the spine of the magazine. And there's more to come!

WHAT'S UP FOR 1995, NEIL?

I thought you'd never ask. As of this writing, several things are going on. For example, we're putting together plans for even more online support than we've ever had before. This not only includes continuing the services that we've been providing to you for a while now, but expanded services on the net, and new services on eWorld.

Many of you already know about our new CD. The MacTech 1-9 CD should be ready by the time you read this. Every article of the magazine, volumes 1-9 has been moved into THINK Reference format with hyperlinks to the Inside Macintosh databases. Initial response to our CD has been overwhelming and very positive. So, if you haven't seen a demo, check it out.

For years now, Apple has put together a "Developer Central" showcase at Macworld. This year, using the best that Apple and MacTech have to offer, we are combining forces to bring the largest Developer Central ever to the show floor. The new Dev Central, sponsored by Apple Computer, Inc. and MacTech Magazine, will take place at Macworld Expo/San Francisco, January 4-7, 1995. Come by and say hello!

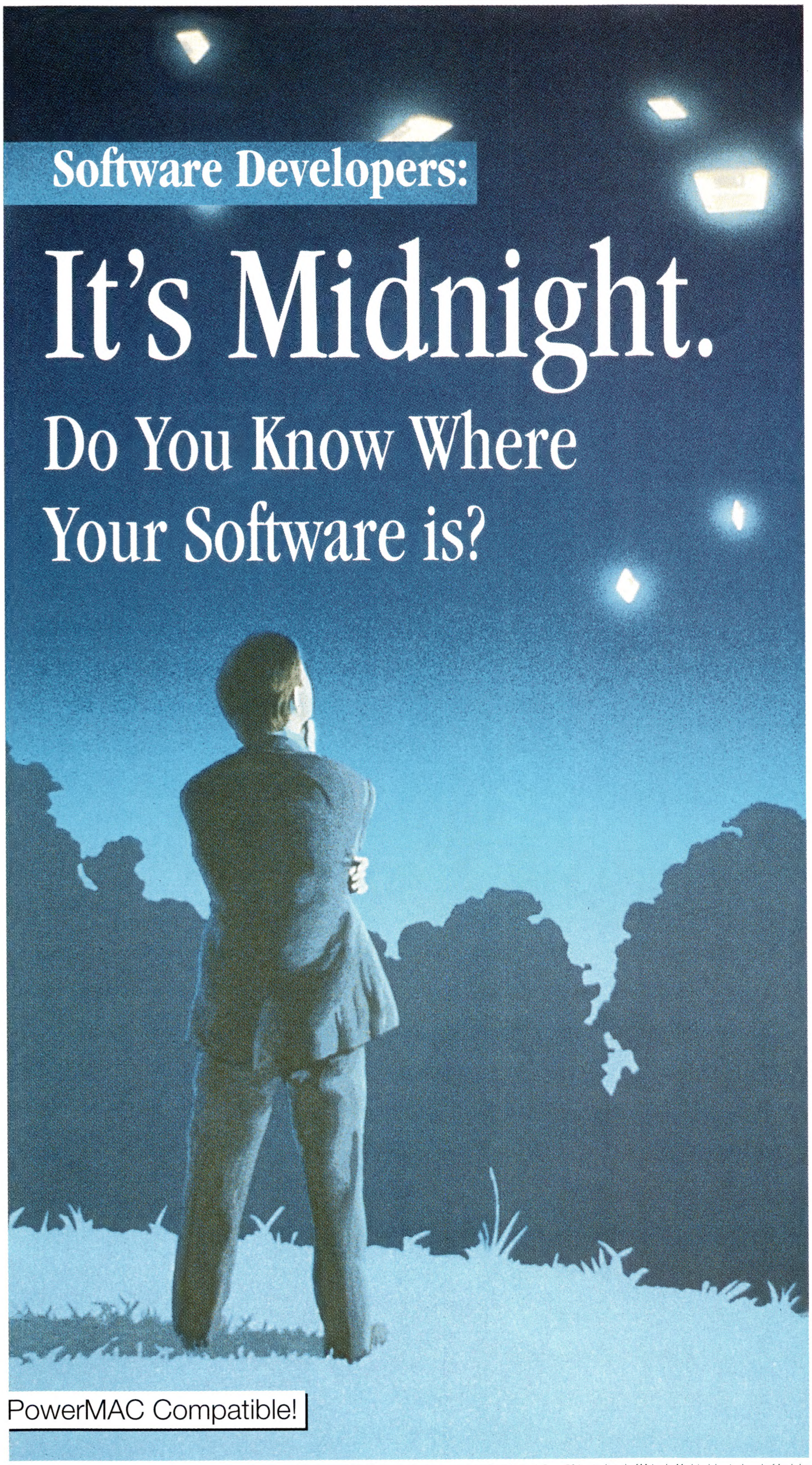
Those of you who were subscribers this summer saw the first ever bound-in CD for MacTech. Well, it's going to happen again. In the January issue, we'll be binding in both OpenDoc and OLE CDs. This issue will have articles to help explain and demonstrate these technologies to you. You can pick up this issue on the newsstand, or you can guarantee that you'll get a copy if you subscribe by November 15th.

Better information. Better tools. Better support. We've been doing a lot at the magazine to enhance our customer service and support. Starting about a year ago, we reached a level of customer service that many of you have been praising regularly. Our goal is to continue improving our support and to help you get more information about what is available out there for you – the Macintosh developer.

You can help! As you can see, we've got a lot going on at the magazine. And I've only told you about the tip of the iceberg. We put a lot of value in what you say to us. We want you to speak up on any topic that you think is relevant – Apple, Microsoft, OpenDoc, MacTech, whatever you want. By letting us know what you think, we can group comments together and let Apple know what developers really want. And, if you want us to help by improving our support of you in some way, let us know that too!

Our goal is for you to be the most productive Macintosh developer that you can be. We're also striving to make sure that there are as many developers for the Mac as possible. In the end, the Macintosh and all of the Macintosh community will thrive and prosper. I'm looking forward to our second decade!

[illegible]



Bringing software into the world is a little like bringing up children. You always know where they start, but you seldom know where they'll end up. These days, with illegal use of software so common, concerned developers have good reason to worry about the products of their labor. That's where MacHASP comes in.

Like a responsible babysitter, MacHASP accompanies your software wherever it goes. With MacHASP there, your software won't run out of control. Without MacHASP, in fact, your software won't run at all.

For developers, MacHASP provides the highest level of security and reliability. For legitimate users, MacHASP is a friendly and transparent solution. Once connected, they won't even feel it's there.

And if your child wants to play with its friends, a single Net-MacHASP lets it run free around a local area network. But always under your supervision and control.

Get serious about software protection. Since 1984, nearly one million HASP keys have enabled thousands of PC & Mac software developers, in more than 60 countries, to protect their software. To find out why MacHASP is considered the best product in the market, order your MacHASP Developer's Kit today.

1-800-223-4277

ALADDIN

The Professional's Choice

North America	**Aladdin Software Security Inc.** Tel: (800) 223 4277, 212-564 5678 Fax: 212-564 3377 E-mail: sales@hasp.com
Intl Office	**Aladdin Knowledge Systems Ltd.** Tel: 972-3-537 5795, Fax: 972-3-537 5796 AppleLink: ALADDIN.KNOW E-mail: aladdin@aladdin.co.il
United Kingdom	**Aladdin Knowledge Systems UK Ltd.** Tel: 0753-622266, Fax: 0753-622262
France	**Aladdin France SA** Tel: 1 40 85 98 85, Fax: 1 41 21 90 56

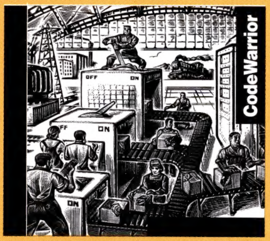
member of

■ **Australia** Conlab 3 8985685 ■ **Benelux** Aladdin Benelux 080 782098 ■ **Czech** Atlas 2 766085 ■ **Chile** Micrologica 2 222 1388
■ **Denmark** Berendsen 39 577100 ■ **Egypt** Zeineldein 2 3604632 ■ **Finland** ID-Systems 0 870 3520 ■ **Germany** CSS 201 278804
■ **Greece** Unibrain 1 6856320 ■ **Italy** Partner Data 2 26147380 ■ **Japan** Athena, 3 58 213284 ■ **Korea** Dae-A 2 848 4481
■ **New Zealand** Training , 4 5666014 ■ **Poland** Systherm 61 480273 ■ **Portugal** Futurmatica 1 4116269 ■ **South Africa** D Le Roux, 11 886 4704
■ **Spain** PC Hardware, 3 4493193 ■ **Switzerland** Opag 61 7169222 ■ **Taiwan** Teco 2 555 9676 ■ **Turkey** Mikrobeta 312 467 7504

We will not be emulated.

Metrowerks CodeWarrior. The only native development environment for the Power Macintosh is here.

Bronze For 68K Mac	**$ 99**
Gold For Power & 68K Mac	**$399**



Metrowerks CodeWarrior is a screamer.
Experience RISC technology. Compile 200,000 lines a minute on the Power Macintosh 8100.

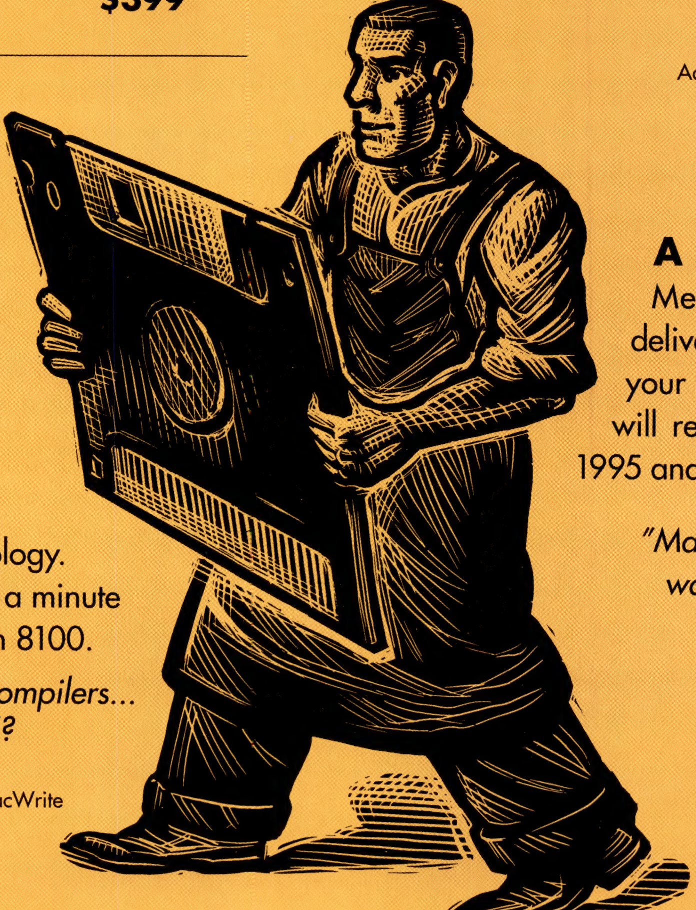
"Great company, fast compilers... how can you beat that"?

Lee Richardson
Development Manager, MacWrite
Claris Corporation

Metrowerks CodeWarrior is awesome.
Native C++ and C compilers for the Power Macintosh. Native C++, C and Pascal compilers for the 68K Macintosh. 68K Macintosh-hosted PowerPC compilers. Power Macintosh and 68K Macintosh source-level debuggers. A new Power Macintosh and 68K Macintosh application framework–Metrowerks PowerPlant. ToolServer, SourceServer and other developer utilities from Apple Development Products. It's all here.

"Without the Metrowerks PowerPC compiler it would be virtually impossible to develop Adobe Illustrator for the Macintosh on the PowerPC."

Don Melton
Software Engineering Leader
Adobe Illustrator for Power Macintosh
Adobe Inc.



Metrowerks CodeWarrior. A new way to buy.
Metrowerks CodeWarrior delivers 3 times a year. With your purchase of CW4, you will receive CW5 in January 1995 and CW6 in May 1995.

"Mac developers have been waiting a long time for an environment like Metrowerks CodeWarrior."

Stanley Crane
General Manager R&D,
cc: Mail Division
Lotus Development Corp.

metrowerks

GETTING STARTED

By Dave Mark, MacTech Magazine Regular Contributing Author

The Required Apple Events

Supporting them is easy

In last month's column, I told you about a new class library we'll be using as the basis for much of the software featured in this column. I hope to bring you some Sprocket material starting next month. In the meantime, we've gotten a lot of requests for some code that handles the four required Apple events. That's what this month's column is all about.

The Required Apple Events

In the old days (before System 7), when a user double-clicked on a document, the Finder first looked up the document's creator and type in its desktop database to figure out which application to launch. It then packaged information about the document (or set of documents if the user double-clicked on more than one) in a data structure, launched the appropriate application and passed the data structure to the application. To access this data structure, the application called the routine `CountAppFiles()` (to find out how many documents it needs to open or print) then, for each one, it called `GetAppFiles()` (to get the information necessary to open the file) and either opened or printed the file.

That model is no longer supported in the Power Mac interface files. It's also way outdated. While existing applications which use the AppFiles method are supported as a compatibility feature of the system, any new code written these days should support the current Apple event scheme. When you mark your application as a modern, with-it application supporting high-level events, launching an application follows a different path. When a user opens a document, the Finder still uses the file's creator and type to locate the right application to launch, but that's where the similarity ends. Once the application is launched, the Finder sends it a series of Apple events.

- If the application was launched by itself, with no documents, the Finder sends it an Open Application Apple event. This tells the application to do its standard initialization and assume that no documents were opened. In response to an **Open Application** Apple event, the application will usually create a new, untitled document.

- If a document or set of documents were used to launch the application, the Finder packages descriptions of the documents in a data structure know as a descriptor, adds the descriptor to an **Open Document** Apple event, then sends the event to the application. When the application gets an Open Document event, it pulls the list of documents from the event and opens each document.

"These events are the four required Apple events ... your application must handle these events."

- If the user asked the Finder to print, rather than open a document or set of documents, the Finder follows the exact same procedure, but sends a **Print Document** Apple event instead of an Open Document event. In response to a Print Document Apple event, the application prints the document rather than opening it.

- Finally, if the Finder wants an application to quit (perhaps the user selected **Shutdown** from the **Special** menu) it sends the application a **Quit Application** Apple event.

When the application gets a quit application event, it does whatever housekeeping it needs to do in preparation for quitting, then sets the global flag that allows it to drop out of the main event loop and exit.

These events are the four required Apple events. As the name implies, your application must handle these events. There are a couple of other situations where your application might receive one of these events.

For starters, any application can package and send an Apple event. If you own a recent copy of QuicKeys, you've got everything you need to build and send Apple events. If you install AppleScript, you can use the Script Editor to write scripts that get translated into Apple events. If you make your application recordable (so that the user can record your application's actions using the Script Editor, or any other Apple event recording application) you'll wrap all of your program's actions in individual Apple events. This means that when the user selects **Open** from the **File** menu, you'll send yourself an Open Document Apple event. If the user quits, you'll send yourself a Quit Application event.

In addition to the events described above, there are other situations in which the Finder will send you one of the four required Apple events. If the user double-clicks on (or otherwise opens) one of your application's documents, the Finder will package the document in an Open Document Apple event and send the event to your application. The same is true for the Print Document Apple event.

The user can also drag a document onto your application's icon. If your application is set up to handle that type of document, your application's icon will invert and, when the user let's go of the mouse, the Finder will embed the document in an Open Document Apple event and send the event to your application. Note that this technique can be used to launch your application or to request that your application open a document once it is already running.

You probably know this technique as Drag-and-Drop. Actually, Drag-and-Drop is the new System software that lets you drag data between applications. The real name for this technique is drop-launching. There's a little coverage of drop-launching in *Inside Macintosh: More Macintosh Toolbox*.

Apple Event Handlers

Apple events are placed in an event queue, much like the events you already know, love, and process, such as mouseDown, activateEvt, and updateEvt. So far, the events you've been handling have all been low-level events, the direct result of a user's actions. The user uncovers a portion of a window, an updateEvt is generated. The user clicks the mouse button, a mouseDown is generated.

Apple events, on the other hand, are known as high-level events, the result of interprocess communication instead of user-process communication. As you process events retrieved by WaitNextEvent(), you'll take action based on the value in the event's what field. If the what field contains the constant updateEvt, you'll call your update handling routine, etc. If the what field contains the constant kHighLevelEvent, you'll pass the event to the routine AEProcessAppleEvent():

```
switch ( eventPtr->what )
{
  case mouseDown:
    HandleMouseDown( eventPtr );
    break;
  case keyDown:
  case autoKey:
    theChar = eventPtr->message & charCodeMask;
    if ( (eventPtr->modifiers & cmdKey) != 0 )
      HandleMenuChoice( MenuKey( theChar ) );
    break;
  case updateEvt:
    DoUpdate( eventPtr );
    break;
  case kHighLevelEvent:
    AEProcessAppleEvent( eventPtr );
    break;
}
```

AEProcessAppleEvent() passes the event to the Apple event handler you've written specifically for that event. To handle the four required events, you'll write four Apple event handlers. You'll install the handlers at initialization time by passing the address of each handler (in the form of a universal-procedure-pointer) to AEInstallEventHandler(). AEProcessAppleEvent() calls your handler for you automatically. Once your handler is installed your work is done.

This Month's Program: AEHandler

This month's program, AEHandler, provides a skeleton you can use to add the required Apple events to your own programs. We'll start off by creating the AEHandler resources. Create a folder called AEHandler in your development folder. Launch ResEdit and create a new file called AEHandler.π.rsrc in the AEHandler folder.

1) Create an MBAR resource with an ID of 128 containing the MENU IDs 128, 129, and 130.

2) Use the specs in Figure 1 to create three MENU resources with IDs of 128, 129, and 130.

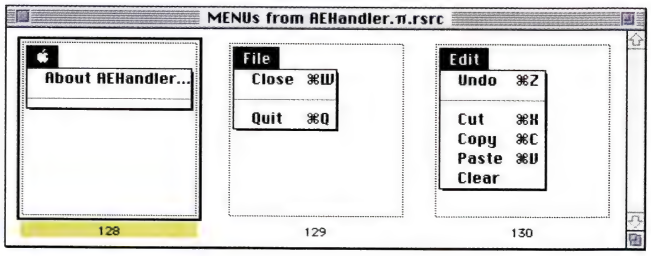


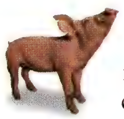
Figure 1. The three MENUs used by AEHandler.

3) Create a WIND resource with an ID of 128, having a top of 41, a left of 3, a bottom of 91, and a right of 303. Use the

Protect your software with Sentinel and watch your profits go hog wild.

Nothing can eat away profits like a herd of pigs. That's why farmers in 47 countries manage swine production with PigCHAMP® software. And that's why PigCHAMP protects their products with Sentinel®, from Rainbow Technologies.

As a developer, you know the importance of profit margins. If you sell software, you're probably losing revenue to piracy. Protect with Sentinel and get all the revenue you deserve.



"Illegal duplication can drive the cost of software sky high. Sentinel protection lets us sell our products inexpensively throughout the world, while serving our customers better," explains PigCHAMP's Steve Abrams.

SUPPORT TECHNOLOGY SENTINEL RELIABILITY QUALITY

Over 11,000 developers protect their software with Sentinel. For DOS, Windows, NT, OS/2, Macintosh, UNIX, or any platform Sentinel is the worldwide standard in software protection.

Implementing Sentinel protection is quick and easy. And only Sentinel delivers the most advanced technology and highest quality every time, all backed by the industry's largest technical staff.

So fatten up your software sales. Protect with confidence, protect with Sentinel. Call for a Developer's Kit today.

1-800-852-8569

SENTINEL

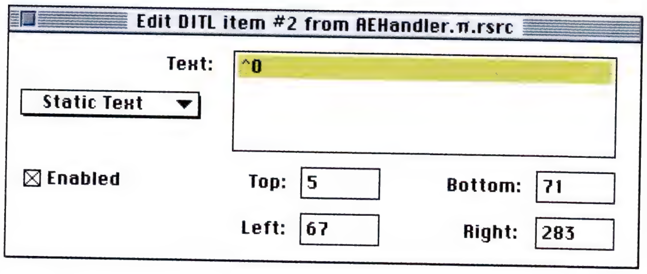
Securing the future of software

RAINBOW TECHNOLOGIES

Rainbow Worldwide HQ: Irvine, CA. Phone 714-454-2100 • Fax 714-454-8557 • Rainbow North Carolina: Phone 800-843-0413 • International offices in U.K., France and Germany • Distributors located worldwide.

standard document proc (left-most in a ResEdit editing pane).

4) Copy the standard error alert from one of our previous programs. If you don't have one, create an ALRT with an ID of 128, a top of 40, left of 40, bottom of 156, and right of 332. Next, create a DITL with an ID of 128 and two items. Item 1 is an OK button with a top of 86, a left of 219, a bottom of 106, and a right of 279. Item 2 is a static text item just like the one shown in Figure 2.


Edit DITL item #2 from AEHandler.π.rsrc

Text: ^0

Static Text

☒ Enabled

Top:	5	Bottom:	71
Left:	67	Right:	283

Figure 2. The static text item for the error alert.

That covers the standard resources. Next come the resources that link specific document types to our application and that tie a set of small and large icons to our application. The Finder uses these resources to display an icon that represents our application in different situations (a large icon when the app is on the desktop, a small icon to display in the right-most corner of the menu bar when our app is front-most). The Finder uses the non-icon resources to update its desktop database.

5) Create a new BNDL resource with a resource ID of 128. When the BNDL editing window appears in ResEdit, select **Extended View** from the **BNDL** menu. This gives you access to some additional fields.

6) Put your application's four-byte signature in the **Signature:** field. Every time you create a new application, you'll have to come up with a unique four-byte string unique to your application. To verify that the signature is unique, you'll need to send it to the AppleLink address DEVSUPPORT. If you don't have a signature handy, feel free to use mine (DM=a). I've registered it for one of my applications but since it's not an application I distribute, you won't run into any conflicts.

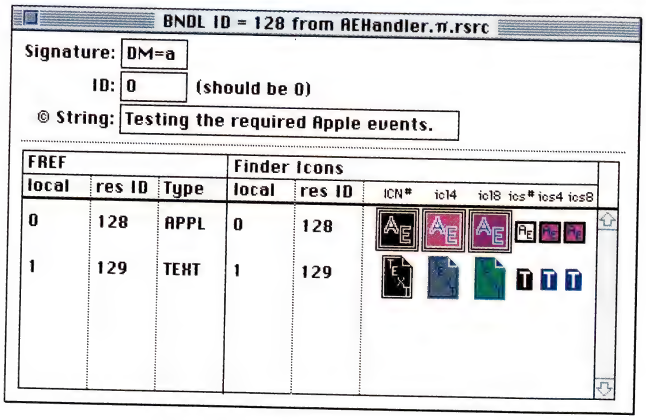
To register your creator/file types with Apple, you'll need to fill out a special request form and send it in to the AppleLink address DEVSUPPORT. Here's where you can find the form:

- AppleLink™ network (Developer Support:Developer Services:Developer Technical Support:Often used DTS Forms:Creator/File Type Form)
- Your monthly Developer CD (Apple Information Resources:Developer Info Assistant:Developer forms:Creator/File Type form)
- Internet: ftp.apple.com (IP address 130.43.2.3) using the path /ftp/dts/mac/registration/.

7) Put 0 in the BNDL's **ID** field.

8) Put a copyright string in the **© String** field. This string will appear in the Finder's get info window for your application.

9) Select **New File Type** from the **Resource** menu. Use the specifications in Figure 3 to fill out the information for the APPL file type. This ties the first row of icons to the application itself. To edit the icons, double-click on the icon area and ResEdit will open an icon family editing window.


BNDL ID = 128 from AEHandler.π.rsrc

Signature: DM=a

ID: 0 (should be 0)

© String: Testing the required Apple events.

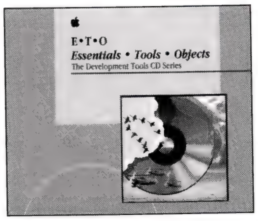
FREF			Finder Icons		
local	res ID	Type	local	res ID	ICN# icl4 icl8 ics# ics4 ics8
0	128	APPL	0	128	
1	129	TEXT	1	129	

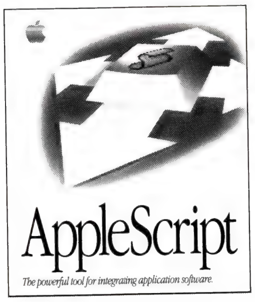
Figure 3. The AEHandler BNDL resource.

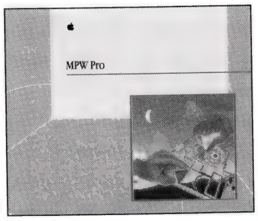
10) Back in the BNDL editing window, select **New File Type** again to add a second row of file types to the BNDL window. This time use the specs in Figure 3 to fill out the info for files of type TEXT. By doing this, we've told the finder that files with the signature 'DM=a' and of type 'TEXT' belong to the application AEHandler. Once again, double-click on the icon family to edit the individual icons.

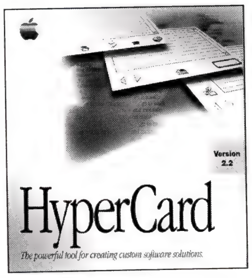
If your application will support file types belonging to other applications, create file type entries in the BNDL resource for them as well, but don't edit the icons – leave them blank.

Streamline your application development with powerful tools from Apple.®









E.T.O.
Essentials•Tools•Objects

E.T.O. is a deluxe collection of core programming tools for Macintosh® developers. It contains an integrated set of development environments, compilers, debuggers, application frameworks, and testing tools that you can use to streamline your application development. E.T.O. is sold by annual subscription.

E.T.O.:
Essentials • Tools • Objects
Complete New Subscriber Package
M0895LL/C $1095.00

Annual E.T.O. Subscription Renewal (3 issues)
R0076LL/B $400.00

AppleScript™

AppleScript is a powerful and versatile tool that makes it easy to integrate the functionality of scriptable applications into a single, seamless, custom solution.

AppleScript Software Development Toolkit v.1.1
R01752Z/C $199.00

MPW® Pro

MPW Pro contains a comprehensive set of tools you can use to develop Macintosh software in C, C++, or assembly language for 68K-based and Power PC-based Macintosh systems. It also includes MacApp,® an object-oriented application framework for accelerating application development.

MPW Pro
R0505LL/C $495.00

MPW Development System to MPW Pro upgrade
R0582LL/A $295

HyperCard® 2.2

HyperCard is an ideal development tool for a wide range of applications. HyperCard works like a deck of electronic index cards. You can easily integrate 24-bit color, PICTs, QuickTime™ and AppleScript to create custom software solutions and manage information.

HyperCard 2.2
M2365LL/A (U.S.) $99.00
M2365Z/A (Int'l.) $99.00

To order or request your free Apple developer tools catalog, call APDA®:
U.S.: 1-800-282-2732
Canada: 1-800-637-0029
International: 716-871-6555

MTM 1294

In addition, be aware that the Finder uses the file type entries to determine what files can drop launch your application. Right now, the Finder will only let you drop launch files with the signature 'DM=a' and of type 'TEXT' on AEHandler. To make AEHandler respond to all file types, create a new file type entry with the file type '****'. Don't edit the icons – leave them blank.

11) Save your changes and close the resource file.

12) In ResEdit, create a new resource file called test.text.

13) Select **Get Info for test.text** from the File menu.

14) When the info window appears, set the file's type to **TEXT** and its creator to whatever signature you used (if you used mine, it's **DM=a**).

That's it. Save your changes, quit ResEdit, and let's create the project.

Creating the AEHandler Project

Pick your favorite development environment and create a new project called AEHandler.π, saving it in the AEHandler folder. Immediately edit your project type info (In THINK C, select **Set Project Type...** from the **Project** menu. In Code Warrior, pick the Project icon in the Preferences dialog). Set the project's creator to the creator you used (mine was 'DM=a'). Next, be sure to set the High-Level Event Aware flag in the SIZE resource flags. *If you don't do this, the Apple Event Manager won't call your handlers!*

Next, add either MacTraps or MacOS.lib to your project. Then, create a new source code file, save it as AEHandler.c and add it to your project. Here's the source code:

```
#include <GestaltEqu.h>
#include <AppleEvents.h>

#define kBaseResID       128
#define kErrorALRTid     128
#define kWINDResID       128

#define kVisible         true
#define kMoveToFront     (WindowPtr)-1L
#define kSleep           60L
#define kNilFilterProc   0L
#define kGestaltMask     1L
#define kKeepInSamePlane false

#define kWindowStartX    20
#define kWindowStartY    50

#define mApple           kBaseResID
#define iAbout           1

#define mFile            kBaseResID+1
#define iClose           1
#define iQuit            3
```

Globals

```
Boolean    gDone;
short      gNewWindowX = kWindowStartX,
           gNewWindowY = kWindowStartY;
```

Functions

```
void    ToolboxInit( void );
void    MenuBarInit( void );
void    AEInit( void );
void    AEInstallHandlers( void );
pascal OSErr DoOpenApp( AppleEvent *event,
                        AppleEvent *reply, long refcon );
pascal OSErr DoOpenDoc( AppleEvent *event,
                        AppleEvent *reply, long refcon );
OSErr  CheckForRequiredParams( AppleEvent *event );
pascal OSErr DoPrintDoc( AppleEvent *event,
                        AppleEvent *reply, long refcon );
pascal OSErr DoQuitApp( AppleEvent *event,
                        AppleEvent *reply, long refcon );
void       OpenDocument( FSSpec *fileSpecPtr );
WindowPtr  CreateWindow( Str255 name );
void       DoCloseWindow( WindowPtr window );
void    EventLoop( void );
void    DoEvent( EventRecord *eventPtr );
void    HandleMouseDown( EventRecord *eventPtr );
void    HandleMenuChoice( long menuChoice );
void    HandleAppleChoice( short item );
void    HandleFileChoice( short item );
void    DoUpdate( EventRecord *eventPtr );
void    DoError( Str255 errorString );
```

main

```
void main( void )
{
  ToolboxInit();
  MenuBarInit();
  AEInit();

  EventLoop();
}
```

ToolboxInit

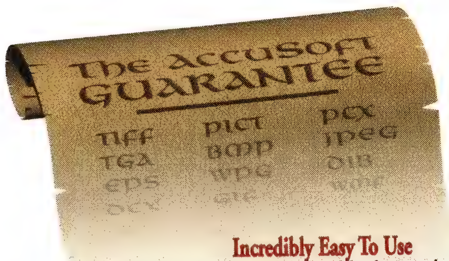
```
void ToolboxInit( void )
{
  InitGraf( &qd.thePort );
  InitFonts();
  InitWindows();
  InitMenus();
  TEInit();
  InitDialogs( 0L );
  InitCursor();
}
```

MenuBarInit

```
void MenuBarInit( void )
{
  Handle      menuBar;
  MenuHandle    menu;

  menuBar = GetNewMBar( kBaseResID );

  if ( menuBar == NULL )
    DoError( "\pCouldn't load the MBAR resource..." );

  SetMenuBar( menuBar );

  menu = GetMHandle( mApple );

  AddResMenu( menu, 'DRVR' );

  DrawMenuBar();
}
```

AEInit

```
void AEInit( void )
{
  OSErr  err;
  long feature;
```



Now get the fastest imaging toolkit with the most platforms, the most formats, and the only guarantee.

AccuSoft is now the recognized provider of the highest quality imaging toolkits in the world. Our performance, compatibility, ease-of-use, service, and our AccuSoft Image Guarantee have earned us this distinction. Our libraries save you hundreds of hours of work and provide your application with a unique advantage: Guaranteed Format Support.

Incredibly Easy To Use

Our toolkits are so easy to use that it should take you less than an hour to add complete image import, export, conversion, display, printing and scanning support to your application. If you need to work at a lower level for special situations, we have functions for that, too!

Story Behind The Guarantee

Having offered this Guarantee for over two years, we have collected several thousand (weird, but perfectly valid) images. Nowadays we see very few problem images. If we do, the image is usually corrupted or invalid, yet we are able to provide solutions for these images as well.

Performance Tuned By Experts

We at AccuSoft have an unyielding commitment to achieving the highest performance possible for all our products. This, of course, means that your products will also achieve greater performance.

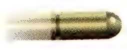
Support For All Platforms

AccuSoft has versions for all major platforms including DOS, 32-bit DOS, Clipper™, Foxpro™, Windows™, Visual Basic®, Windows NT, OS/2, Chicago, Macintosh®, and UNIX. We can also port to other systems upon request.

Complete Compression

All forms of compression are supported including JPEG, Group III, Group IV, Packbits, LZW, Huffman, and more. Read any image format, then convert to any other format-with only two function calls!

AccuSoft is the Fastest!



Features, Features, Features

We provide a complete set of functions for printing, scanning, display, image processing and image handling. All printers and TWAIN scanners are supported with simple function calls. The image printing engine produces high quality output regardless of the printer. The display engine provides high-speed (up to 50 times faster than Windows) and high quality display for all types of images and display modes. Our image processing functions include zoom, pan, scroll, invert, rotate, resize, sharpen, blur, contrast, brightness, palette optimization, color reduction, matrix convolutions and much more.

Visual Basic® Versions

We have special versions for Visual Basic that provide all the power of the DLL in the form of a Custom Control. We even have the world's only 32-bit VBX for Windows 3.1. *Extremely Fast!*

Pro Gold Versions

Our Pro Gold libraries represent the most advanced performance and features available in the world. Pro Gold versions are available for Windows 3.1 (no special hardware or drivers required), Mac, UNIX and others. When your application demands unbeatable performance, go with Pro Gold.

Call Now To Order **800-525-3577** *Risk-Free 30-day money back guarantee on all 16-bit products.*

ACCUSOFT IMAGE GUARANTEE

**Guaranteed to read all raster images in existence in the supported formats. If you can find a valid image we don't read, send it to us and we will make it work.*

AccuSoft
High Performance Imaging

AccuSoft Corporation, 112 Turnpike Road, Westborough, MA 01581 TEL:(508) 898-2770 FAX: (508) 898-9662

```
  err = Gestalt( gestaltAppleEventsAttr, &feature );

  if ( err != noErr )
    DoError( "\pError returned by Gestalt!" );

  if (   !( feature
         & ( kGestaltMask << gestaltAppleEventsPresent ) ) )
    DoError("\pThis configuration does not support Apple events...");

  AEInstallHandlers();
}
```

AEInstallHandlers

```
void AEInstallHandlers( void )
{
  OSErr         err;

  err = AEInstallEventHandler( kCoreEventClass,
                          kAEOpenApplication,
          NewAEEventHandlerProc( DoOpenApp ), 0L, false );

  if ( err != noErr )
    DoError( "\pError installing 'oapp' handler..." );

  err = AEInstallEventHandler( kCoreEventClass,
          kAEOpenDocuments,
          NewAEEventHandlerProc( DoOpenDoc ), 0L, false );

  if ( err != noErr )
    DoError( "\pError installing 'odoc' handler..." );

  err = AEInstallEventHandler( kCoreEventClass,
          kAEPrintDocuments,
          NewAEEventHandlerProc( DoPrintDoc ), 0L, false );

  if ( err != noErr )
    DoError( "\pError installing 'pdoc' handler..." );

  err = AEInstallEventHandler( kCoreEventClass,
          kAEQuitApplication,
          NewAEEventHandlerProc( DoQuitApp ), 0L, false );

  if ( err != noErr )
    DoError( "\pError installing 'quit' handler..." );
}
```

DoOpenApp

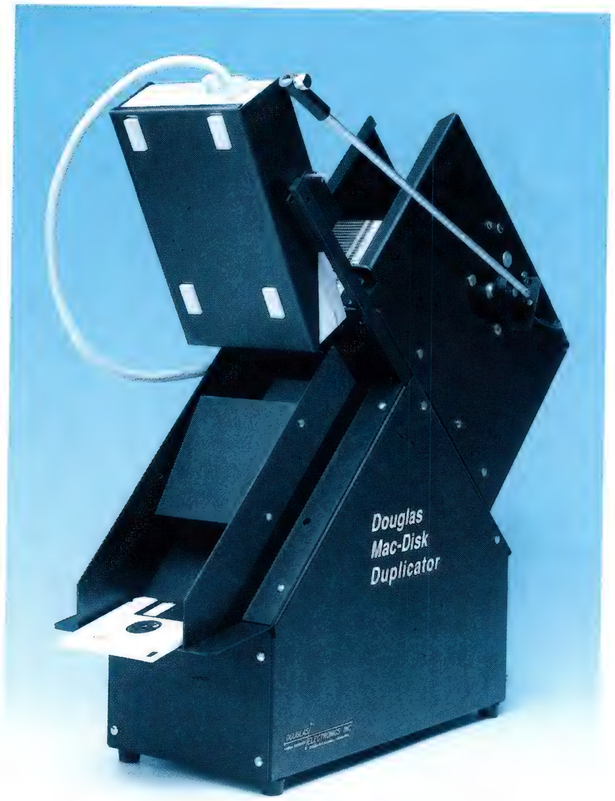
```
pascal OSErr DoOpenApp( AppleEvent *event,
                        AppleEvent *reply, long refcon )
{
  OpenDocument( nil );

  return noErr;
}
```

DoOpenDoc

```
pascal OSErr DoOpenDoc( AppleEvent *event,
                        AppleEvent *reply, long refcon )
{
  OSErr     err;
  FSSpec    fileSpec;
  long      i, numDocs;
  DescType returnedType;
  AEKeyword  keywd;
  Size    actualSize;
  AEDescList docList = { typeNull, nil };

  // get the direct parameter–a descriptor list–and put
  // it into docList
```

Douglas MacDisk Duplicator

The low-cost solution to disk duplication!



The simple, low cost, no frills MacDisk Duplicator is the answer to your software duplication needs. No other duplicator can match its ease of use and efficiency for the price. All you need to operate the duplicator is an Apple Macintosh computer. From there, you simply plug the duplicator's disk drive into the Macintosh, load the diskettes into the input hopper and get back to your other work. A short while later, your disks will be ready for labeling and distribution. Disk duplication couldn't be any easier or more economical than with the MacDisk Duplicator from Douglas Electronics. The MacDisk Duplicator runs on any Macintosh with an external floppy port.

Pricing:

$1,500 without disk drive

$1,850 with FDHD disk drive

For More Information:

DOUGLAS
ELECTRONICS
INC.

2777 Alvarado Street San Leandro, California 94577
[510] 483-8770 FAX 483-6453

```
    err = AEGetParamDesc( event, keyDirectObject,
                          typeAEList, &docList);

    // check for missing required parameters
    err = CheckForRequiredParams( event );
    if ( err )
    {
        // an error occurred:  do the necessary error handling
        err = AEDisposeDesc( &docList );
        return err;
    }

    // count the number of descriptor records in the list
    // should be at least 1 since we got called and no error
    err = AECountItems( &docList, &numDocs );

    if ( err )
    {
        // an error occurred:  do the necessary error handling
        err = AEDisposeDesc( &docList );
        return err;
    }

    // now get each descriptor record from the list, coerce
    // the returned data to an FSSpec record, and open the
    // associated file
    for ( i=1; i<=numDocs; i++ )
    {
        err = AEGetNthPtr( &docList, i, typeFSS, &keywd,
                    &returnedType, (Ptr)&fileSpec,
                    sizeof( fileSpec ), &actualSize );

        OpenDocument( &fileSpec );
    }

    err = AEDisposeDesc( &docList );

    return err;
}
```

CheckForRequiredParams

```
OSErr  CheckForRequiredParams( AppleEvent *event )
{
    DescType returnedType;
    Size    actualSize;
    OSErr     err;

    err = AEGetAttributePtr( event, keyMissedKeywordAttr,
            typeWildCard, &returnedType,
            nil, 0, &actualSize);

    if ( err == errAEDescNotFound ) // you got all the required
                                    //parameters
      return noErr;
    else
      if ( err == noErr )           // you missed a required parameter
        return errAEParamMissed;
      else                          // the call to AEGetAttributePtr failed
        return err;
}
```

DoPrintDoc

```
pascal OSErr DoPrintDoc( AppleEvent *event,
                         AppleEvent *reply, long refcon )
{
  return noErr;
}
```

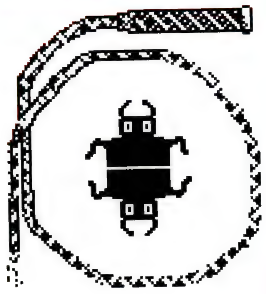
DoQuitApp

```
pascal OSErr DoQuitApp( AppleEvent *event,
                        AppleEvent *reply, long refcon )
{
  SysBeep( 20 );
  gDone = true;

  return noErr;
}
```

Gives you the Information to *Program* *your* *Best!*

The Debugger V2 & MacNosy

by Steve Jasik



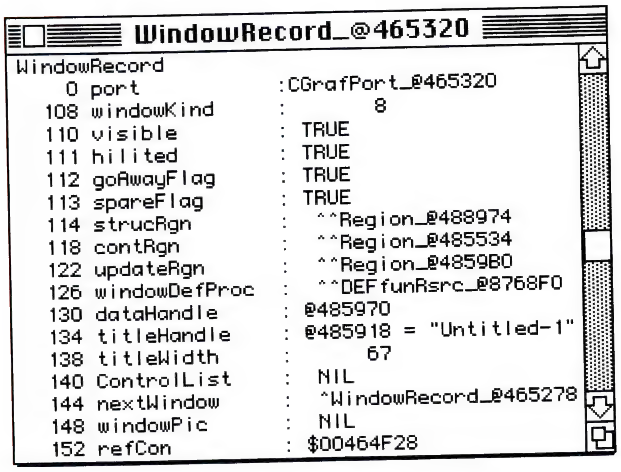
The Debugger is a low and high-level symbolic Debugger that runs in a full multi-window Macintosh environment. You can trace program execution, view the values of variables, etc. ***of both 68K and PowerPC programs***.

MacNosy is a global interactive disassembler that enables one to recover the source code of any Mac application, resource file or the ROM.

When you compare features of the different debuggers, note that *only one* has all the below features to help you get your job done, and *only one* has MacNosy to help you debug any program in a full system (6.0x or System 7.x) environment symbolically!

It is the *only* debugger to use the MMU to protect your CODE resources and the rest of the system from the program you are debugging. With MMU Protection you can find errors when they happen, not millions of instructions later! (Macintoshs with 68030 CPUs only).

The Debugger is the debugger of choice at: Adobe, Aldus, Claris, Electronic Arts, Kodak, Metrowerks, etc.



```
WindowRecord_@465320
WindowRecord
   0 port            :CGrafPort_@465320
 108 windowKind      :        8
 110 visible         :  TRUE
 111 hilited         :  TRUE
 112 goAwayFlag      :  TRUE
 113 spareFlag       :  TRUE
 114 strucRgn        :   ^^Region_@488974
 118 contRgn         :   ^^Region_@485534
 122 updateRgn       :   ^^Region_@4859B0
 126 windowDefProc   :   ^^DEFfunRsrc_@8768F0
 130 dataHandle      :  @485970
 134 titleHandle     :  @485918 = "Untitled-1"
 138 titleWidth      :        67
 140 ControlList     :    NIL
 144 nextWindow      :    ^WindowRecord_@465278
 148 windowPic       :    NIL
 152 refCon          :  $00464F28
```

An example of a structured data display window

Its Features Include:

- **Symbolic Debugging** of *any* Macintosh program, ROM, or *code resource* (DRVRs, XCMDs, INITs, PDEFs, 4DEXs ..)
- **Source level debugging for Metrowerks & MPW** compiled programs (C++, C, Pascal, Fortran, ...), and an Incremental Build System with instant Link for superfast development.
- **Object Inspector for MacApp 3 programs**
- **Source level debugging of Think C™ projects**
- **Includes a program (CoverTest) to interactively do Code Coverage analysis for SQA testing, etc.**
- **Simultaneous Symbolic debugging of multiple "tasks"**
- **Fast Software Watchpoint** command to find clobbered variables
- **Sophisticated error check algorithms** such as Trap Discipline (Argument Checking), Handle Zapping, Heap Scramble and Heap Check to detect program errors before they become disasters
- **Structured display** of data (hypertext) with user definable structures while debugging
- **Conditional breakpoints** to help filter out redundant information
- **Continuous Animated Step Mode** to watch your program execute instruction by instruction
- **Detailed symbolic disassembly for both 680x0 and PowerPC** with symbol names, labels, cross ref maps, – make it possible to ferret out the secrets of the ROM, etc.
- **"Training Wheels"** for the PowerPC disassembler to help you learn the opcodes

The Debugger V2 & MacNosy: $350
Runs on all Macs. Call For Group prices or Updates. Visa/MC Accepted.

Available from: Jasik, APDA, Frameworks or ComputerWare (800-326-0092).

Jasik Designs • **343 Trenton Way, Menlo Park, California 94025** • **(415) 322-1386**
Internet: macnosy@netcom.com • **Applelink: D1037**

OpenDocument

```
void OpenDocument( FSSpec *fileSpecPtr )
{
  WindowPtr  window;

  if ( fileSpecPtr == nil )
    window = CreateWindow( "\p<Untitled>" );
  else
    window = CreateWindow( fileSpecPtr->name );
}
```

CreateWindow

```
WindowPtr  CreateWindow( Str255 name )
{
  WindowPtr  window;
  short      windowWidth, windowHeight;

  window = GetNewWindow( kWINDResID, nil, kMoveToFront );

  SetWTitle( window, name );

  MoveWindow( window, gNewWindowX, gNewWindowY,
              kKeepInSamePlane );

  gNewWindowX += 20;
  windowWidth = window->portRect.right
              - window->portRect.left;

  if ( gNewWindowX + windowWidth > qd.screenBits.bounds.right )
  {
    gNewWindowX = kWindowStartX;
    gNewWindowY = kWindowStartY;
  }

  gNewWindowY += 20;
  windowHeight = window->portRect.bottom
               - window->portRect.top;

  if ( gNewWindowY + windowHeight >
                         qd.screenBits.bounds.bottom )
  {
    gNewWindowX = kWindowStartX;
    gNewWindowY = kWindowStartY;
  }

  ShowWindow( window );
  SetPort( window );

  return window;
}
```

DoCloseWindow

```
void DoCloseWindow( WindowPtr window )
{
  if ( window != nil )
    DisposeWindow( window );
}
```

EventLoop

```
void EventLoop( void )
{
  EventRecord  event;

  gDone = false;
  while ( gDone == false )
  {
    if ( WaitNextEvent( everyEvent, &event, kSleep, nil ) )
      DoEvent( &event );
  }
}
```

DoEvent

```
void DoEvent( EventRecord *eventPtr )
{
  char  theChar;

  switch ( eventPtr->what )
  {
    case mouseDown:
      HandleMouseDown( eventPtr );
      break;
    case keyDown:
    case autoKey:
      theChar = eventPtr->message & charCodeMask;
      if ( (eventPtr->modifiers & cmdKey) != 0 )
        HandleMenuChoice( MenuKey( theChar ) );
      break;
    case updateEvt:
      DoUpdate( eventPtr );
      break;
    case kHighLevelEvent:
      AEProcessAppleEvent( eventPtr );
      break;
  }
}
```

HandleMouseDown

```
void HandleMouseDown( EventRecord *eventPtr )
{
  WindowPtr    window;
  short      thePart;
  long      menuChoice;

  thePart = FindWindow( eventPtr->where, &window );

  switch ( thePart )
  {
    case inMenuBar:
      menuChoice = MenuSelect( eventPtr->where );
      HandleMenuChoice( menuChoice );
      break;
    case inSysWindow :
      SystemClick( eventPtr, window );
      break;
    case inGoAway:
      if ( TrackGoAway( window, eventPtr->where ) )
        DoCloseWindow( window );
      break;
    case inContent:
      SelectWindow( window );
      break;
    case inDrag :
      DragWindow( window, eventPtr->where, &qd.screenBits.bounds );
      break;
  }
}
```

HandleMenuChoice

```
void HandleMenuChoice( long menuChoice )
{
  short  menu;
  short  item;

  if ( menuChoice != 0 )
  {
    menu = HiWord( menuChoice );
    item = LoWord( menuChoice );

    switch ( menu )
    {
      case mApple:
        HandleAppleChoice( item );
        break;
      case mFile:
        HandleFileChoice( item );
        break;
    }
    HiliteMenu( 0 );
  }
}
```

HandleAppleChoice

```
void HandleAppleChoice( short item )
{
  MenuHandle appleMenu;
  Str255     accName;
  short      accNumber;

  switch ( item )
  {
    case iAbout:
      SysBeep( 20 );
      break;
    default:
      appleMenu = GetMHandle( mApple );
      GetItem( appleMenu, item, accName );
      accNumber = OpenDeskAcc( accName );
      break;
  }
}
```

HandleFileChoice

```
void HandleFileChoice( short item )
{
  switch ( item )
  {
    case iClose:
      DoCloseWindow( FrontWindow() );
      break;
    case iQuit:
      gDone = true;
      break;
  }
}
```

DoUpdate

```
void DoUpdate( EventRecord *eventPtr )
{
  WindowPtr  window;

  window = (WindowPtr)eventPtr->message;

  BeginUpdate(window);
  EndUpdate(window);
}
```

DoError

```
void DoError( Str255 errorString )
{
  ParamText( errorString, "\p", "\p", "\p" );

  StopAlert( kErrorALRTid, kNilFilterProc );

  ExitToShell();
}
```

Running AEHandler

Save your code, then build AEHandler as an application (In MetroWerks, just run the application) and then run it. An untitled window should appear. If it didn't, go back and check your SIZE resource to make sure the High-Level-Event Aware flag is set.

As you look through the code, you'll see that the untitled window is created by the Open Application handler. Now double click on the file test.text. A window titled test.text should appear. This window was created by the Open Documents handler.

With AEHandler still running, go into the Finder and select **Shutdown** from the **Special** menu. The Finder should bring AEHandler to the front and send it a Quit Application Apple event. Our Quit Application handler beeps once then sets gDone to true. When you quit normally, you won't hear this beep.

Till Next Month

Hopefully, next month will bring the first Sprocket column. Till then, read up on the Apple Event Manager. Play around with the AEHandler code. Add some code to the Open Document handler to open the specified file and display info about the file in its window (maybe the file's size). If you are interested in learning more about Apple events, check out the first few chapters of *Ultimate Mac Programming*, due out in January from IDG Books. See you next month…

Apprentice

Mac Source Code CD-ROM

Over 450 megabytes of up-to-date Mac-only source code in C, C++, and Pascal. Libraries, MPW tools, languages, utilities, information, demos, and much more! $35 including shipping. Add $5 for shipping outside of the U.S. and Canada. VISA, MC, AMEX, and Discover gladly accepted.

Celestin Company, 1152 Hastings Ave, Port Townsend, WA 98368
800 835 5514 • 206 385 3767 • 206 385 3586 fax
Internet: celestin@olympus.net • CompuServe: 71630,650

FREE INSTALLER!

To: Mac Developers & Product Managers

From: MindVision (Creators of Stacker for Macintosh)

Subject: <u>Free Copy of DEVELOPER VISE 3.0</u>

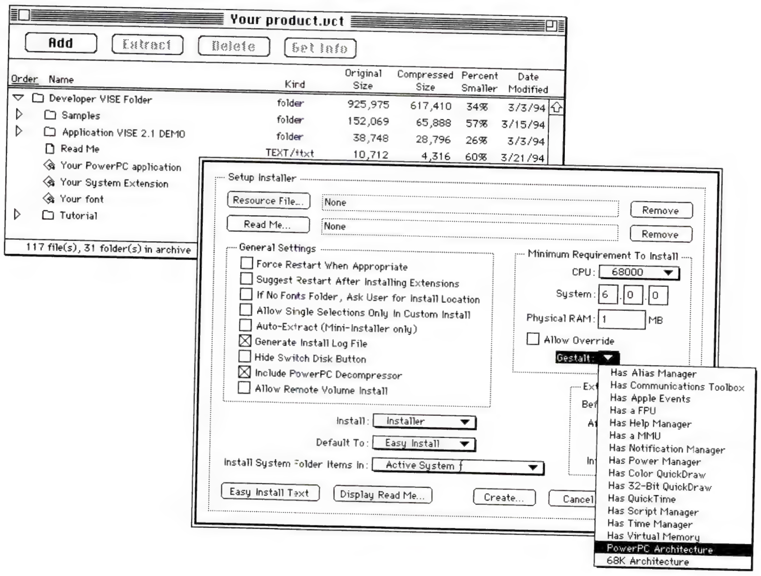
Message: We've got a great new installer for you. Here's your chance to try it for free. No risk, no obligation, no hassle. Call today, don't delay!

Full PowerPC Support

Integrated compression

No programming required

Fully graphical interface



Order Name	Kind	Original Size	Compressed Size	Percent Smaller	Date Modified
Developer VISE Folder	folder	925,975	617,410	34%	3/3/94
Samples	folder	152,069	65,888	57%	3/15/94
Application VISE 2.1 DEMO	folder	38,748	28,796	26%	3/3/94
Read Me	TEXT/ttxt	10,712	4,316	60%	3/21/94
Your PowerPC application					
Your System Extension					
Your font					
Tutorial					

117 file(s), 31 folder(s) in archive

Setup Installer

Resource File...: None

Read Me...: None

General Settings

- ☐ Force Restart When Appropriate
- ☐ Suggest Restart After Installing Extensions
- ☐ If No Fonts Folder, Ask User for Install Location
- ☐ Allow Single Selections Only In Custom Install
- ☐ Auto-Extract (Mini-Installer only)
- ☒ Generate Install Log File
- ☐ Hide Switch Disk Button
- ☒ Include PowerPC Decompressor
- ☐ Allow Remote Volume Install

Minimum Requirement To Install

CPU: 68000

System: 6 . 0 . 0

Physical RAM: 1 MB

☐ Allow Override

Install: Installer

Default To: Easy Install

Install System Folder Items In: Active System f

Call (402) 477-3269

IF YOU PREFER, USE E-MAIL. APPLELINK, AOL: MINDVISION • COMPUSERVE 70253,1437

Join the growing list of companies who use our VISE technology: Adobe, Apple, CE Software, Claris, CompuServe, DeltaPoint, MacroMedia, Radius, Stac Electronics, Symantec, WordPerfect, and many more.

MindVision Software

MindVision Software 840 South 30th St., Suite C, Lincoln, Nebraska 68510
Voice: (402) 477-3269 Fax: (402) 477-1395 AppleLink, AOL: MindVision

VISE COMPRESSION TECHNOLOGY

IMPROVING THE FRAMEWORK

By Richard Gillam, GE Information Services

Better Window Management in MacApp

Plugging some holes in MacApp windows

The little things count. This article doesn't aim to present any Higher Truths of Object-Oriented Computing, or to show you how to solve third-order differential equations using the THINK Class Library, or anything like that. Instead, I want to share with you a little class I created a while back to plug up a few holes in MacApp 3.0.1's window-management code.

What's Wrong With MacApp's Window Stuff?

Some things don't quite live up to their press. As with printing in MacApp ("It's only a couple lines of code" only if you're doing really basic, no-frills printing), so it is with window management in MacApp, only not quite as bad. I found two significant deficiencies in MacApp's window-management code and one feature I wanted to add.

The first feature was MacApp's window-staggering code. If you want the initial positions of your document windows to march down the screen and to the right as you successively open windows, rather than opening directly on top of each other, you can set the window's `fMustStagger` flag in the view template resource. But if you do that without implementing a staggering routine of your own, you'll run into two big problems. The first is a truly brain-dead positioning algorithm. MacApp keeps a global variable that contains a count of how many staggered windows it has opened. To figure out where to open the next one, it just multiplies the standard staggering offset by the number of windows that have already been opened. That's it. This means that if you've opened ten windows and then closed all of them, the next window you open will still be halfway down the screen. Ugly.

"If you've opened ten windows and then closed all of them, the next window you open will still be halfway down the screen. Ugly."

Worse, there's a bug in their algorithm. The algorithm is smart enough to sense when you've marched off the edge of the monitor and move you back to the upper left-hand corner, but they didn't do it quite right. If you have two monitors and the auxiliary monitor is placed to the right of the main monitor (the one with the menu bar), you can run into a situation where new windows open with their initial position straddling monitors, a definite human-interface no-no. This is either because they don't take multiple monitors into account or because they only check to see if the window has gone off the bottom of the screen and not if it's gone off the right-hand side.

Rich Gillam is a conservatory-trained orchestral percussionist masquerading as a MacApp developer. When not playing the drums, singing, or playing with his cat, Rich puts food on the table by "bringing good things to life" at GE Information Services, Inc. He says that cats sound different when you play them depending on what kind of food they've been eating off of the table.

Simply the best **GUI Building/Event Managing** libraries

Tools Plus™ 2.5

Tools Plus gives you the routines you need to create a professional looking user interface. Then we make it work. It's that simple.

- For THINK C and Symantec C/C++ (5.0.4 and later) or THINK Pascal
- Over 170 high-powered "set and forget" routines that automate and enhance: event handling, windows, the tool bar, floating palettes, cursors, buttons, picture buttons, scroll bars, menus (pull-down, hierarchical and pop-up), list boxes, fields, Edit menu, clipboard, Dynamic Alerts, and more...
- Easy to learn and easy to use
- Substantial code reduction
- Dramatic code simplification
- Significantly less debugging
- System 6 and 7 compatible
- For novice, intermediate and advanced programmers
- Runs fast; needs little memory or disk space
- Safer than toolbox routines
- No royalties

OK

Water's Edge Software

Box 70022 • 2441 Lakeshore Road West
Oakville, Ontario • Canada L6L 6M9
Phone: (416) 219-5628
CompuServe: 73424,2507
Internet: 73424.2507@CompuServe.com

Tools Plus for C/C++ or Pascal only $149 US or $199 US for both.
(We accept VISA and American Express. Add $10 for shipping.)

Language
○ C/C++
○ Pascal
● Both

Size: 9 10 ● 12 14 18 24 36

☒ Saves Time
☒ Saves Money
☒ Easy to Use

Free Evaluation Kit:

CompuServe GO MACDEV,
C & Pascal library, file name: TP252.SEA

AppleLink Software Sampler/Software Collection/Programmer Tools/Tools Plus

Disk also available by mail.

[To see what I'm talking about, comment out the line that begins "`RegisterStdType("TBetterWindow"...)`" in the demo program and compile and run it. On a two-monitor machine, if you open enough windows, several of them will hang off on the right-hand monitor. They won't move back to the upper left-hand corner until they march off the bottom.]

The other feature I wanted to improve was the window-zooming code. Here, MacApp does take multiple monitors into account, but doesn't take the contents of the window itself into account. The human-interface guidelines say that clicking the zoom box of a window should make it just large enough to hold its contents (if that'll fit on the screen), but no larger. MacApp's default window-zooming algorithm always zooms the window to full-screen size.

Finally, I needed to add the ability to save and restore window positions.

The TBetterWindow Class

You'll find source for a class designed to solve that above problems and a demo program called BetterWindow in the usual source code locations. The program is basically the Nothing application beefed up with the addition of the `TBetterWindow` class. I made the default view wider so that you could see the multiple-monitor bug, and I added the ability to make the view itself larger and smaller. You may want to follow along in the demo program as I go through these examples.

It would be nice if we could improve the positioning and zooming algorithms by adding a behavior to a regular `TWindow`. Unfortunately, we can't, because we have to override too many routines which are specific to `TWindow` and not available to override from `TBehavior`. So I had to subclass `TWindow` and create a class called `TBetterWindow`. Fortunately, for certain classes, MacApp provides a way to use a subclass instead of standard class. In your application's initialization code (probably in your application class's I method), call `RegisterStdType()`. My call to `RegisterStdType()` looks like this:

```
RegisterStdType("TBetterWindow", kStdWindow);
```

`kStdWindow` tells MacApp you're providing a class name to use when opening a standard window, and the string parameter is the class name to use instead of `TWindow`. Now anywhere you use `TWindow`, `TBetterWindow` will be substituted.

You can probably safely use `TBetterWindow` all the time, even in cases (such as dialog boxes) where you don't need its features. But if you want more control over when specifically you use `TBetterWindow`, you need to specify it in all of the view hierarchy resources where you intend to use it.

Smart Window Positioning

To control how a window is staggered, you override `TWindow::SimpleStagger()`. `TWindow::Open()` calls this routine any time the window being opened has its `fMustStagger` flag set. This routine positions and resizes the window as appropriate. It takes two parameters: the amount to offset each window from the previous one, and a count of the staggered windows opened so far. We don't use the count parameter.

The basic idea behind this routine is that every new window opens offset by the specified amount from the window right behind it. But we've added two important refinements: 1) If that position will run partially off the screen or straddle two monitors, we don't use it, but instead move back toward the upper left-hand corner of the monitor containing the largest part of the previous active window. 2) If the position where we want to open the window is already occupied by another

Apple's Installer 4.0*

ScriptGen Pro 2.0

Install your product using the familiar interface of Apple's Installer (the same program your customers use to install System software).

New in ScriptGen Pro 2.0
- **Installer 4.0 Support**
- **Developer Hooks**
- **Folder Copies**
- **Interface & Speed Improvements**

"Its intuitive interface makes creating Installer scripts blissfully simple and quick"
MacTech Magazine

Functional Demo Available:
AppleLink: Third Party Demos
AOL: Developers Forum
CIS: Developers Forum
eWorld

ScriptGen Pro: $169
InstallerPack: $219

STEPUP SOFTWARE
7110 Glendora Avenue
Dallas, Texas 75230
214-360-9301
214-360-0127 fax

ScriptGen Pro is a trademark of StepUp Software. Apple is a registered trademark of Apple Computer, Inc.
* Requires additional licensing

window (i.e., another window has its upper left-hand corner where we were going to put this window's upper left-hand corner), then move down and to the right until we find an unoccupied position.

```
pascal void TBetterWindow :: SimpleStagger(
                            CPoint   delta,
                            short&   /*count*/)

  {
    CPoint         ulhc(7, 20);
    CPoint         newPos;
    WindowPtr        frontWindow;
    CRect            temp;
    CRect            monitorRect;
    CPoint         localUlhc;
    VRect            frame;

    // take into account the menu bar when determining
    // where to put the first window
    ulhc.v += *(short*)MBarHeight;

    // find the front window and take note of its monitor
    // rectangle
    frontWindow = MAGetActiveWindow();
    GetMonitorInfo(frontWindow, monitorRect);

    // if there is no front window, the new window will go
    // in the upper left-hand corner of the main monitor,
    // defined above
    if (frontWindow == NULL) {
      newPos = ulhc;
      localUlhc = ulhc;
    }

    // if there is a front window, the new one will be
    // offset from it by the value of "delta" and the upper
    // left-hand corner we use for positioning will be that
    // of the monitor containing the most of the current
    // front window (notice we have to adjust for the menu
    // bar size)
    else {
      temp = (**((WindowPeek)frontWindow)->contRgn)
                .rgnBBox;
      newPos = temp[topLeft];
      newPos += delta;
      localUlhc = ulhc;
      if (monitorRect[topLeft] != gZeroPt) {
        localUlhc += monitorRect[topLeft];
        localUlhc.v -= *(short*)MBarHeight;
      }
    }
    this->MakePositionUnique(newPos, localUlhc, delta,
              monitorRect);
    this->Locate(VPoint(newPos), false);
  }
```

Note the extra code that is required to take the menu bar and multiple monitors into account. Most of this code is fairly self-explanatory, but it is worth noting that "ulhc" is initialized to (7, 20) to give adequate clearance around the window. The position the Mac OS uses is actually the upper left-hand corner of the window's content area, so we have to leave room for the title bar (hence the 20), and it looks a little better if we also leave some room on the left-hand side (hence the 7).

There are two refinements that can be added to the above algorithm. First, for performance reasons, it's a good idea to take advantage of the window's `fStaggered` flag:

```
// if fStaggered is true, then we've already done all this
if (this->fStaggered)
  return;

// otherwise, go ahead and set the flag now so we won't go
// through all this the next time
this->fStaggered = true;
```

Second, if the window is a text-editing window, it's nice to take advantage of all available screen real estate by elongating the window after you've positioned it:

```
if ( /*I'm a text-editing window*/ ) {
  this->GetFrame(frame);
  frame.bottom = monitorRect.bottom - 3;
  this->Resize(frame.GetSize(), false);
}
```

If you elongate the window afterwards, the window can take up all available vertical space, no matter how much there is. It

Power Mac and Macintosh Developers:

FIND OUT FAST WHAT'S GOING ON IN MEMORY

THE MEMORY MINE™

- See memory allocation in any open heap at a glance.
- Easily spot memory leaks.
- Flags heap corruption when it happens.
- Works with source level debugger to let you find memory problems fast.
- Stress applications on the fly with Purge, Compact, and Zap.
- Allocate memory at will for precise stress testing.
- Log heap data - easily document heap status over time.
- No need for source code: nothing inserted in code; no patches to the system.
- Works with 24-bit, 32-bit, and modern memory managers.

For Macintoshes with 68020 or better. Requires System 7.0 or later.

only $99 US
Order now from Adianta, Inc.
Phone: (408)354-9569 • FAX: (408)354-4292
AppleLink:ADIANTA • AOL:Adianta • Internet:adianta@aol.com
For VISA, MC, or American Express orders by mail, fax, or Applelink, please include name, address, card number, expiration date, and phone number or email address.
Also available through the MacTech Mail Order Store.

for more information contact

Adianta, Inc. • 2 N. Santa Cruz Ave. #201 • Los Gatos, CA 95030

also means you can define the window's size in your view template as the minimum size of the window. The positioning algorithm will move the window back to the upper left-hand corner when you're too far down to show this minimum height.

Of course, much of the real work of the above routine is being done in subroutines. `GetMonitorInfo()` is used several different places to retrieve the bounding rectangle of the monitor containing the largest part of the specified window.

```
pascal void TBetterWindow :: GetMonitorInfo(
                                WindowPtr   frontWindow,
                                CRect&      monitorRect)
  {
    GDHandle    monitor;
    GrafPtr     wMgrPort;
    TWindow*    frontTWindow;

// if we don't have Color QuickDraw, we can't have
// multiple monitors, so by definition our entire
// universe is defined by the Window Manager port's
// portRect
    if (!gConfiguration.hasColorQD) {
      GetWMgrPort(wMgrPort);
      monitorRect = wMgrPort->portRect;
      return;
    }

// if there are no open windows, use the main monitor
    if (frontWindow == NULL) {
      monitor = GetMainDevice();
      monitorRect = (**monitor).gdRect;
    }

// otherwise, use a method in TWindow to find out which
// monitor contains more of the frontmost window and
// then get info for it
    else {
      frontTWindow = WMgrToWindow(frontWindow);
      if (frontTWindow == NULL) {
        monitor = GetMainDevice();
        monitorRect = (**monitor).gdRect;
      }
      else {
        monitor = frontTWindow->
                    GetMaxIntersectedDevice(monitorRect);
        monitorRect.top -= *(short*)MBarHeight;
      }
    }
  }
```

In most cases, we can take advantage of a MacApp routine called `TWindow::GetMaxIntersectedDevice()` to find out which monitor a window is on. `GetMonitorInfo()` helps us by taking care of the situation where we don't have Color QuickDraw (in which case we can use the Window Manager port's portRect, since there can only be one monitor), and the situation where there isn't actually an open window to use (in which case we get the rectangle of the main monitor).

The job of find an unoccupied position for the new window is done by `TBetterWindow::MakePositionUnique()`.

```
pascal void TBetterWindow :: MakePositionUnique(
                                CPoint&         newPos,
                                CPoint          ulhc,
                                CPoint          delta,
                                const CRect&  monitorRect)
  {
    CPoint      workPos(newPos);
    CPoint      tempUlhc;

// for as long as the proposed position is on the
// screen and another window occupies this position,
// keep advancing down and to the right until we
// either march off the screen or find an unoccupied
// position
    while (monitorRect.Contains(workPos) &&
               !IsPositionUnique(workPos))
      workPos += delta;

// if we've marched off the screen or if the unoccupied
// position won't allow the whole window to be on the
// screen, try again starting at the monitor's upper
// left-hand corner
    if (!monitorRect.Contains(workPos) ||
               !WillWindowFit(workPos, monitorRect)) {
      if (newPos != ulhc) {
        workPos = ulhc;
        if (!IsPositionUnique(workPos)) {
          MakePositionUnique(workPos, ulhc, delta,
                                          monitorRect);

// if we can't find an unoccupied position that
// will hold the window after starting at the
// upper left-hand corner of the monitor, try
// again, but start halfway between the first
// two window positions on the monitor (the test
// here causes recursive calls to fall out if
// they don't find an acceptable position)
          if (workPos == ulhc) {
            workPos += CPoint(delta.h / 2, delta.v / 2);

            if (!IsPositionUnique(workPos)) {
              tempUlhc = workPos;
              MakePositionUnique(workPos, tempUlhc,
                          delta, monitorRect);

// if that doesn't work either, give up and
// just put the window in the upper left-
```

Cross-Platform Object Database Engine

neoAccess™

Easy to Use NeoAccess is a set of easy to understand C++ classes that extend the capabilities of **PowerPlant 1.0**, **MacApp 3.1** and the **THINK Class Library 2.0** on the Macintosh and **MFC 2.5**, **ObjectWindows 2.0** and **zApp 2.1** in Intel-based environments. Suddenly persistent C++ objects are first-class citizens that can be organized and accessed naturally using an API designed for minimum visible complexity. And to make things even easier, NeoAccess comes complete with ***full source code***.

Full Featured ***NeoAccess 3.0 is now available.*** It includes support for threads, blobs, part lists, iterators, swizzlers, temporary objects, multiple indices on a class, a powerful relational object selection mechanism, duplicate keys, a versatile streams I/O model and incredible performance. NeoAccess has a huge storage capacity with no database administration to worry about. And databases are standard document files with your application icon on them. They are also binary-compatible across platforms.

High Performance The use of binary trees and an efficient object caching mechanism means very fast access to objects and reduced file I/O. Only objects of immediate interest need to be in memory, so your applications can be much smaller, even when accessing huge databases.

Proven Hundreds of commercial and in-house C++ developers are using NeoAccess to develop powerful CAD/CAM, multi-media, document management, data visualization, accounting, CD-ROM titles and many other kinds of applications. NeoAccess is what your organization needs to realize its potential.

Affordable NeoAccess's best feature is its price. Our development tools have always been the best value available. The NeoAccess Developer's Toolkit sells for just $749 per developer with ***no runtime licensing fees***. It includes ***full source code***, numerous sample applications and 350+ pages of documentation.

Native **PowerPC** support too!

neo•logic
In order to survive, you need to persist.

No Runtime Licensing Fees:
Use NeoAccess in all your development for one low price!

1450 Fourth St. • Suite 12 • Berkeley • CA • 94710 • (510) 524-5897 • AppleLink: NeoLogic

```
// hand corner of the monitor
                  if (workPos == tempUlhc)
                    workPos = ulhc;
                }
              }
            }
          }
        else
          workPos = newPos;
      }

// return the new position
    newPos = workPos;
  }
```

The basic idea here is that we check the current position of the window. If it's occupied or part of the window dangles off the monitor, move down and to the right by the offset amount and try again. If the upper left-hand corner of the window marches off the screen, start over again (calling ourselves recursively), but start at the upper left-hand corner of the monitor. If that doesn't work either, try again (with another recursive call), with the first spot being half the offset value from the upper left-hand corner of the screen. If that doesn't work either (at which point some twenty-odd positions will have been tried), we just give up and put the window in the upper left-hand corner of the screen.

I'll grant that the giving-up part isn't real cool, but it should only happen under extreme conditions, and there is no easy way to modify this algorithm to reuse used positions (we'd have to count the windows in each position, which is painful). We could go hog-wild with the staggering (after all, there are some 15 valid positions between two windows offset from each other by the standard offset amount). I didn't do this either, but it would be a good way of handling it if you often find yourself in a state where there are too many windows open for this algorithm to support.

`MakePositionUnique()` relies on another routine, `IsPositionUnique()`, to determine when it's found a usable position. `IsPositionUnique()` in turn relies on a routine called `WillWindowFit()` to determine whether the whole window can fit at the specified position without hanging off the screen or straddling monitors:

```
pascal Boolean TBetterWindow :: IsPositionUnique(
                                CPoint      pos)
  {
    CWMgrIterator   iter;
    WindowPtr         aWindowPtr;
    CRect             windowRect;

    for (aWindowPtr = iter.FirstWMgrWindow();
              iter.More();
              aWindowPtr = iter.NextWMgrWindow())
    if (((WindowPeek)aWindowPtr)->visible)
          &&  IsDocumentWindow(aWindowPtr)) {
      windowRect = (**((WindowPeek)aWindowPtr)
                        ->contRgn).rgnBBox;
```

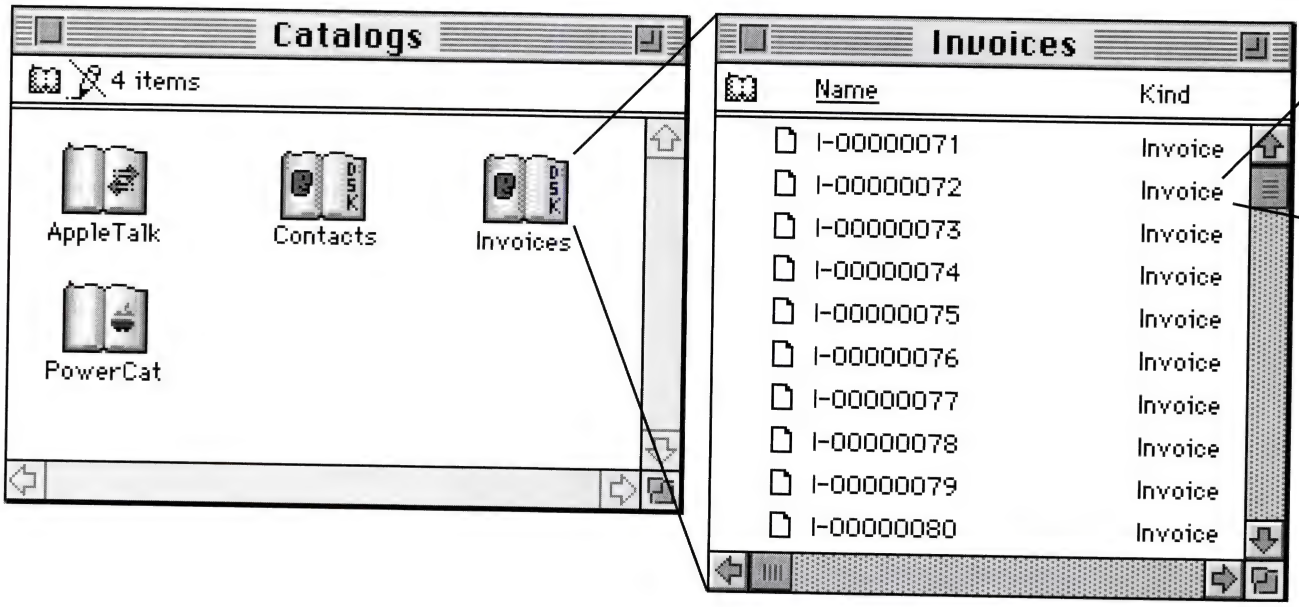
Databases in the Finder?

Unfortunately, most users must interact with relational databases at the SQL level or behind a front-end that can take months to develop.

select invoice_num,invoice_date,cust_id,cust_name from invoices,customers where invoice_id = 112;printall;

Why not view and edit your data at the Finder level? Announcing **Cataloger**™.

Cataloger brings a whole new way of working with databases to the Macintosh. The Finder™ has always provided a view of our desktop files, why not database records too?



Name	Kind
I-00000071	Invoice
I-00000072	Invoice
I-00000073	Invoice
I-00000074	Invoice
I-00000075	Invoice
I-00000076	Invoice
I-00000077	Invoice
I-00000078	Invoice
I-00000079	Invoice
I-00000080	Invoice

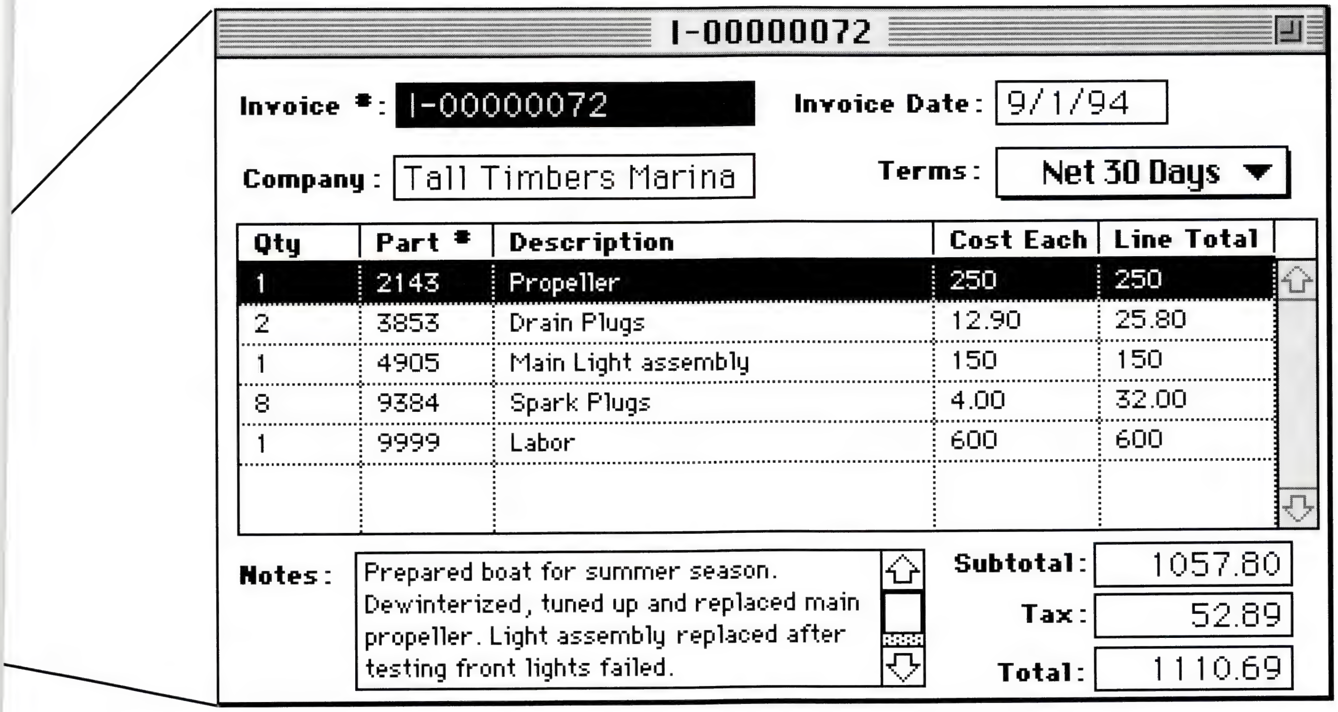
Apple's **O**pen **C**ollaborative **E**nvironment places a catalog icon at the Finder. Open the icon up and you'll see a window like the one above. The AppleTalk catalog shows nodes on the network. The PowerCat catalog shows users and groups in a PowerShare Server.

The Contact and Invoices catalogs (for example) can show information from **4D Server**, **DAL/DAM** , **Oracle** and **Sybase** databases! If your needs are more customized, you can write AppleScript handlers to provide the contents of your catalog.

The Catalog Manager (an AOCE API) allows for **C**atalog **S**ervice **A**ccess **M**odules (CSAMs) to be created that interact with external sources.

The Finder and other applications make requests through the Catalog Manager and Cataloger does the rest. There are no limits to the amount of data brought back.

Templates provide the ability to view and edit individual records. AOCE Templates are designed with **Template Constructor™.** This powerful application let's you easily design forms to view data coming from several sources, control behavior with C and AppleScript, and view row/column data with **TableIt!™** - a powerful data display tool.



I-00000072

Invoice #: I-00000072 **Invoice Date:** 9/1/94

Company: Tall Timbers Marina **Terms:** Net 30 Days

Qty	Part #	Description	Cost Each	Line Total
1	2143	Propeller	250	250
2	3853	Drain Plugs	12.90	25.80
1	4905	Main Light assembly	150	150
8	9384	Spark Plugs	4.00	32.00
1	9999	Labor	600	600

Notes: Prepared boat for summer season. Dewinterized, tuned up and replaced main propeller. Light assembly replaced after testing front lights failed.

Subtotal: 1057.80

Tax: 52.89

Total: 1110.69

Access information and catalog definitions are stored in and protected by the AOCE **Keychain**. Your users can have one username and password for a variety of data sources!

Need more horsepower? **Database Scripting Kit™** provides AppleScript access to all of the connection information and data sources. Write AppleScript routines that send queries and parse results from within your own application.

Cataloger/Template Constructor includes:

Database Scripting Kit™ with connectors for 4D Server, DAL/DAM, Oracle, Sybase and others.

DSK Cataloger™ CSAM

Template Constructor™ with TableIt!™

Several example catalogs/templates and over 100 example AppleScripts.

Complete scriptability, native for Power Macintosh. Drag Manager and Thread Manager support.

How to contact us:

Graphical Business Interfaces, Inc.

Voice: 800-424-7714 or 219-253-8623

Fax: 219-253-7158

E-Mail: steve@gbi.com

GBI. Bringing your data into view.™

```
        if (windowRect[topLeft] == pos)
            return false;
    }
    return true;
}

pascal Boolean TBetterWindow :: WillWindowFit(
                            CPoint          pos,
                            const CRect&  monitorRect)
{
    CRect       frame;

    frame[topLeft] = pos;
    frame[botRight] = pos + this->fSize.ToPoint();

// the insetting here takes into account the frame and
// drop shadow (we don't care about the title bar,
// since we're always going down and to the right
// and we know we started in a position where the title
// bar will work)
    frame.Inset(CPoint(-2, -2));

    return monitorRect.Contains(frame);
}
```

WillWindowFit() is pretty self-explanatory. IsPositionUnique() takes advantage of an internal MacApp class, the CWMgrIterator, to walk the Window Manager's window list to find out whether any other windows already on the screen occupy our proposed position. It's a lot easier to walk the Window Manager's list than to walk MacApp's list.

Smart Window Zooming

To fix the zooming problem, the primary place to tap in is TWindow::GetStandardStateFrame(). There are two TWindow routines: GetStandardStateFrame() and GetUserStateFrame(). The standard state is the system-supplied position and size for the window, a canonical "most natural size" for the window (which is why it's called the "standard state"). The user state is the size the window was before zooming, a state the window probably got into because the user resized or moved it (which is why it's called the "user state"). Resizing the window by dragging the size box always puts the window into the user state. Clicking the window's zoom box toggles between the user and standard states.

We don't need to override GetUserStateFrame() because we're not doing anything to the user state. But the standard state supplied by MacApp is always the full size of the monitor, which is often way too big in at least one dimension for what's actually in the window, so…

```
pascal void TBetterWindow :: GetStandardStateFrame(
                        const VRect& /*boundingRect*/,
                        VRect&          stdFrame)
{
    TView*          targetView;
    TScroller*  targetScroller;
    VRect               targetFrame;
    VRect               windowFrame;
    VPoint          difference;
    GDHandle        monitor;
    CRect               tempMonitorRect;
    VRect               monitorRect;
    VPoint          ulhc(7, 20);

// get the window target view and figure out the
// difference between its current size and the window's
// current size (if the window target view is
// enclosed in a scroller, we're actually interested in
// the size of the scroller here instead [this probably
// won't work right if the window target view isn't the
// only thing in the scroller])
    targetView = (TView*)this->GetWindowTarget();
    targetScroller = targetView->GetScroller(false);
    if (targetScroller == NULL)
        targetView->GetFrame(targetFrame);
    else
        targetScroller->GetFrame(targetFrame);
    this->GetFrame(windowFrame);
    difference = windowFrame.GetSize() -
                targetFrame.GetSize();

// calculate the target view's minimum frame size, and
// calculate a new frame rectangle for the window by
// adding the difference between the window
// size and the old frame size to the new frame size
// and making the window frame that new size
    targetView->CalcMinFrame(targetFrame);
    windowFrame[botRight] = windowFrame[topLeft] +
                targetFrame.GetSize() + difference;
    if (windowFrame.GetLength(hSel) <
            fResizeLimits.left)
        windowFrame.right = windowFrame.left +
                fResizeLimits.left;
    if (windowFrame.GetLength(vSel) <
            fResizeLimits.top)
        windowFrame.bottom = windowFrame.top +
                fResizeLimits.top;

// get the rectangle of the monitor containing the
// largest part of the window (inset it a little to
// leave some slop on the sides and to leave
// room for the menu bar (if we have one) and the
// window's title bar)
```

The professional's database engine

CXBase Pro

for
Macintosh and Power Macintosh

Based on C-Index Pro, the database engine used by **Apple AOS** for their **eWorld** content publishing software, *CXBase Pro* has all the features and power of C-Index Pro, plus additional new capabilities. *CXBase Pro* includes a flexible select module, access to both logical and absolute record numbers, and more.

CXBase Pro is designed as a "pure" database engine. The philosophy behind *CXBase Pro* is to provide a simple, consistent, and powerful API that lets you work in whatever way you choose. *CXBase Pro* is delivered with full source code, giving you maximum flexibility and security. The *CXBase Pro* library provides both multi-keyed data record handling, and flexible BLOB storage where you define the data format. And *CXBase Pro* has been engineered from the ground up for maximum performance.

- **New! Powerful and flexible select module**
  CXBase Pro lets you create multiple kinds of selections, and provides a callback mechanism that allows you to both display the progress, and capture the data as it is selected, giving you full control over the operation of the select routine.

- **New! Access to logical and absolute record numbers**
  You can access data records not only by key, but also by logical and absolute record numbers. This gives you maximum flexibility as to how you retrieve and display your data.

Additional standard features:

- **Unrivaled performance - runs native on the PowerPC!**
- **Royalty-free license**
- **Complete ANSI C source code provided**
- **Data files and code 100% cross-platform**

Full-featured $49 demo available!
CXBase Pro also comes in a demo version. You get the complete manual, and a full-featured library limited only by file size.

For full prices, to order, or to receive more information contact:

TSE International | Taandwarsstr. 51 | 1013 BV Amsterdam, Holland | tel: 31 20 638-6507 | fax: 31 20 620-4933 | AppleLink : TSE.INT

```
// [NOTE: GetMaxIntersectedDevice() will usually take
// the menu bar into account. If it does, the else
// clause below will automatically adjust
// ulhc to take the menu bar into account too.]
    monitor = this->GetMaxIntersectedDevice
                    (tempMonitorRect);
    monitorRect = VRect(tempMonitorRect);
    if (monitorRect[topLeft] == gZeroVPt)
        ulhc.v += *(short*)MBarHeight;
    else
        ulhc += monitorRect[topLeft];
    monitorRect[topLeft] = ulhc;
    monitorRect[botRight] -= VPoint(3, 3);

// does the new window frame fit on that monitor? If
// not, move the window to the upper left-hand corner
// of the monitor
    if (!monitorRect.Contains(windowFrame)) {
        windowFrame += ulhc - windowFrame[topLeft];

// does it fit now? If not, resize it so that as
// much of it as possible does
        if (!monitorRect.Contains(windowFrame)) {
            windowFrame.bottom = Min(windowFrame.bottom,
                        monitorRect.bottom);
            windowFrame.right = Min(windowFrame.right,
                        monitorRect.right);
        }
    }

// and return the result
    stdFrame = windowFrame;
  }
```

The algorithm I'm using here isn't completely general, but it should cover most of the common cases. It works on the theory that when you resize a window by dragging the resize box, the part of the window that actually changes size is the window target view (or more accurately, the scroller containing the window target view). Everything else (scroll bars, other panes, etc.) derives its new size or position from the new size of the window target view's scroller. Obviously, if you have a window for which this isn't true, you'll need to modify this algorithm, but it ought to work for most cases.

At any rate, the routine figures out the optimum size for the window by getting the current size of the window target view's scroller and taking note of the difference between this size and the size of the window's whole content area. It then adds this difference to the size returned by the window target view's `CalcMinFrame()` routine, which is by definition the smallest frame size that will hold the view's contents.

Actually, that's not always true. We've probably all had a programming situation where a view was smaller than its scroller (or window, or containing view, or whatever), but where we wanted to take mouse hits or draw selections and mouse-tracking feedback in the area outside the view. Often, the best way to do this is to enlarge to view so it is always at least as large as its containing view. This, of course, messes up the zooming algorithm.

POSITIONS WANTED

Available Immediately *1-800-736-3577*

Expert 4D Programmers Less than 50¢ per hour!

More than 1,000 hours of development went into 4D Toolkit 2.0. With a one time price of $395.00*, it's like hiring a crack team of 4D programmers at less than 50¢ per hour. Call 1-800-736-3577 to order your copy, or to receive a **free** demo.

4D TOOL **KIT**™

OPtions Computer Consulting
228 Bleecker Street #19
New York, NY 10014
TEL: 212-645-3577
FAX: 212-633-0336

* *upgrade pricing available*

My solution to this is have `CalcMinFrame()` always return the smallest possible view size and do the adjustment in `ComputeFrame()` instead.

```
pascal void TBetterView :: ComputeFrame(
                          VRect&    newFrame)
  {
    VPoint              superViewSize;
    SizeDeterminer     saveHSizeDet, saveVSizeDet;
    Boolean              tryAgain = false;

    inherited::ComputeFrame(newFrame);

// the inherited is sufficient if we're printing or if
// by some weird twist of fate we don't have a
// superview
    if (!fSuperView || gPrinting)
      return;

// otherwise, save off our size determiners (we're
// going to change them)
    saveHSizeDet = fSizeDeterminer[hSel];
    saveVSizeDet = fSizeDeterminer[vSel];
    superViewSize = fSuperView->fSize;

// if this view is smaller than its superview in the x
// direction, change the size determiner to
// sizeSuperView so we make ourselves the same size
    if (fSizeDeterminer[hSel] == sizeVariable &&
               superViewSize[hSel] > newFrame.
               GetLength(hSel)) {
      fSizeDeterminer[hSel] = sizeSuperView;
      tryAgain = true;
    }

// if this view is smaller than its superview in the y
// direction, change the size determiner to
// sizeSuperView so we make ourselves the same size
    if (fSizeDeterminer[vSel] == sizeVariable &&
               superViewSize[vSel] > newFrame.
               GetLength(vSel)) {
      fSizeDeterminer[vSel] = sizeSuperView;
      tryAgain = true;
    }

// if we changed either size determiner, call inherited
// again to make this view at least as big as its
// superview in each direction, and restore the
// real size determiners
    if (tryAgain) {
      inherited :: ComputeFrame(newFrame);
      fSizeDeterminer[hSel] = saveHSizeDet;
      fSizeDeterminer[vSel] = saveVSizeDet;
    }
  }
```

Of course, you can't override `ComputeFrame()` from a behavior either, so it's probably best to put this override of `ComputeFrame()` (possibly controlled by a flag so you don't always have to use it) in a subclass of `TView` that all your view classes descend from.

Up to now, `GetStandardStateFrame()` has just figured the optimum size for the window. If the size will fit on the monitor without moving the window, we're done and we return. If not, we move the window to the upper left-hand corner of the monitor. If it'll fit there, we're done. Otherwise, we shrink the window in each direction until it fits. This way, the window will fill the whole monitor only when all that space is actually filled with information.

We're not done yet. There's one more wrinkle that must be taken into account. Now that the size of the window's standard state is based on the size of the information in the window, the size of the standard state will change when the size of the window's content changes. Since the size of the window can't change when the size of the standard state does, the window must be considered to have transitioned to the user state when its content changes size, just as it does when the user manually resizes the window.

To see what I'm talking about, run the demo program. Open a window. The window is in its user state. Now click on its zoom box. The window is now in its standard state. Now change the size of the window's content. I have fixed it so that hitting U and D make the window's content larger and smaller, respectively (creative, huh?) so you can see this. Make the content bigger by hitting U a couple times. Now if you click on the zoom box again, you don't want to go back to the last user state (the window's original frame size). You want to go to the new standard state size (the current size of the content). If you click the zoom box, you'll see that this is what happens. Now hit D four times to make the view smaller than it originally was. Again, clicking the zoom box should make the window shrink to the current size of the view, not go back to the size it was before you last clicked the zoom box. Again, if you click it, you'll see this is what is does.

To accomplish this, I had to override `TWindow::Zoom()`

as well as TWindow:: GetStandardStateFrame(). This method checks anytime we're in the standard state to make sure the standard state size is still valid:

```
pascal void TBetterWindow :: Zoom(short partCode)
  {
    VRect    curFrame;
    VRect    newStdStateFrame;

    if (partCode == inZoomOut)
      inherited :: Zoom(partCode);
    else {
      this->GetFrame(curFrame);
      this->GetStandardStateFrame(curFrame,
                newStdStateFrame);
// ("curFrame" will be ignored)
        if (curFrame == newStdStateFrame)
          inherited :: Zoom(partCode);
        else {
          if (fProcID & zoomDocProc)
            (*((WStateDataHandle)(((WindowPeek)fWMgrWindow)
                    ->dataHandle)))-> userState =
                    curFrame.ToRect();
          inherited :: Zoom(inZoomOut);
        }
      }
  }
```

If the window is in the user state (partCode == inZoomOut), we can just go ahead and zoom. If the window is in the standard state, we check (by calling GetStandardStateFrame()) to see if the standard state has changed size (because the window target view has changed size and the change affects the window size [going from two feet high to three feet high won't resize the window because two feet is already bigger than the monitor]). If it hasn't, we can call the inherited routine with no further ado. Otherwise, we have to manually transition ourselves to the user state (saving off the window's current size in the WindowRecord's auxiliary data block) and then call the inherited routine as if we had been in the user state all along.

Saving Window Positions

I didn't actually include any window-position-saving code in this article. This is because the act of saving a window position is so application-specific. The way we did it in the application I've been working on doesn't really apply to anybody else's application, and uses proprietary code. But TBetterWindow does include hooks for it.

The basic idea here is that we add two flags to TBetterWindow: fInDefaultPosition and fSavePosition. fInDefaultPosition starts out true when a window is first opened (if its position hasn't been restored from disk). You override MoveByUser(), ResizeByUser(), and ZoomByUser() to set it false so that anytime the user manually moves or resizes the window, you know you have to resize it. fSavePosition starts out false, and is set to true by some outside routine (through the MarkPositionAsDirty() routine) anytime something else has happened that would cause you to save the window position (in case you're saving other stuff, such as sort order or selection position, with the window position).

Now you have a routine called SavePosition(), which is called by an override of TWindow::Close(). It checks these flags and if fInDefaultPosition is false or fSavePosition is true, it saves the window's position. SimpleStagger() now calls a routine called PositionIsSaved() before doing its normal staggering. If PositionIsSaved() returns true, the window's position has been previously saved, and we want to use the saved position rather than deriving a new position algorithmically. So we call RestorePosition(), which restores the window's position, and return without doing anything. The expression of all this in C++ code is included in the source code to the demo program.

#INCLUDE STDDISCLAIMERS.H

As I mentioned at the top of this article, TBetterWindow aims to fix deficiencies in the MacApp 3.0.1 window-handling code. I haven't yet made the switch to MacApp 3.1, so I can't say whether these problems are still there. I suspect they are. I hope, if anyone on the MacApp team is listening, that you adopt something like these solutions in MacApp 3.5.

Meanwhile, I hope someone out there finds these techniques useful. They made a big difference in our application. As always, I welcome your questions, comments, and potshots.

SYMANTEC.™

By Mark B. Baldwin and Michael Hopkins, Symantec Technical Support

This monthly column, written by Symantec's Technical Support Engineers, aims to provide you with information on Symantec products. Each month we cover either a specific application of tools or a "Q&A" list.

Q. *How do I get the menu item "Look up in Think Reference" to work? Whenever I select it, I get an error message "Cannot find Think Reference."*

A. Think Reference is located in the folder Development:Online Documentation. Make an alias of the Think Reference application and move it to the folder named (Tools). Change the alias' name from Think Reference alias to Think Reference.

Make sure to remove the space at the end of the alias name as well.

Q. *Where do I get headers for QuickDraw GX and the Thread Manager?*

A. The headers will be available on the next Developer's Advantage CD. If you are a current subscriber, it will be sent to you automatically. Otherwise, you can order the Developer's Advantage CD by calling 1-800-441-7234.

Q. *When I look at a variable in the Debugger, I see two diamonds and an out of memory error instead of my variable. What can I do to fix this?*

A. The usually happens when the Debugger information has been corrupted. To fix this, you should

- Choose Options from the Edit menu
- Choose Think Project Manager from the hierarchical menu
- Go to the Debugging page and turn on "Store debugging information externally."
- Trash the all the .XSYM files for your project. They are found in the .XSYM folder in your current project folder

Sometimes it is necessary to remove the problematic files and re-add them to the project. In severe cases, use the AppleScripts "Save File as Text" and "Create Project from Text" to recover the project.

Q. *I am having sporadic problems with the Symantec Debugger since I've installed System 7.5. Is there a fix available?*

A. There is a known incompatibility with the Symantec Debugger and System 7.5. We have worked closely with Apple to resolve the problem. A free, on-line patch (version 7.0.4) fixes this problem. The v7.0.4 patches are available on the following services:

AppleLink	Third Parties:Third Parties (P-Z):Symantec Solutions:Software Updates & Solutions:Think C/Symantec C++ Updates:
CompuServe	GO SYMDEVTOOL (Libraries 2 and 11)
AOL	Keyword: SYMANTEC (in our software library)
Symantec BBS....	GO /SymCMac (or GO /ThinkC)
Internet/FTP.......	devtools.symantec.com Macintosh/Updates/ DevTools/ <fileName>

Q. *Why do I get a syntax error in a class declaration in BRClaInfo.h while compiling?*

A. You probably have a file that has a .c extension with a class definition in your project file. When compiling a C++ project, you will get this error. To fix this, rename the offending source file to a .cp extension and add it to your project.

Q. *Why does my window draw with a different size than I expect when I use CDecorator::PlaceNewWindow()?*

A. PlaceNewWindow() resets the sizeRect data member of the window. You can override this functionality in a derived class or not use the CDecorator class to place your window.

Q. *When I am using CSaver, I get a link error for PutObject() and GetObject(). How do I resolve this link error?*

A. To resolve this error, you will need to create a source file

C/C++ without Object Master

Introducing a powerful new way of looking at code development.

Open your eyes to Object Master™, the most innovative programming tool available on the market today. With its powerful editors and intuitive windows, Object Master gives you the unsurpassed freedom to develop code quickly and accurately.

A Real Eye Opener

Use Object Master's unlimited number of browser windows to access code components and display their definitions, ready for editing. All changes you make in the browser windows are automatically displayed throughout the environment, consistently ensuring current and accurate code. With the browser windows, you can also view a class list displaying the hierarchical relationships between classes, and use pre-made templates to create classes and methods.

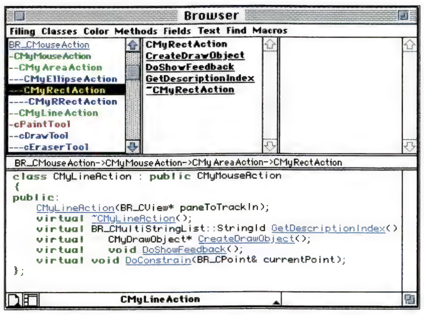
Picture Perfect Code

While the browser windows let you edit specific pieces of code, Object Master's File editor gives you a full-file view of your code, allowing you to quickly perform universal edits.

To help you identify important pieces of code easily, Object Master color codes and formats language elements. With a single keystroke, Object Master will look up parameters for methods or functions contained in the project and paste them directly into your source code.

Improved Insight

Don't worry about the physical location of your code—Object Master parses all files and maintains a data dictionary of project components. The dynamic environment updates your entire project automatically as you edit without compilation!



```
class CMyLineAction : public CMyMouseAction
{
public:
    CMyLineAction(BR_CView* paneToTrackIn);
    virtual ~CMyLineAction();
    virtual BR_CMultiStringList::StringId GetDescriptionIndex()
    virtual    CMyDrawObject* CreateDrawObject();
    virtual    void DoShowFeedback();
    virtual void DoConstrain(BR_CPoint& currentPoint);
};
```

True browser windows let you see code like never before.

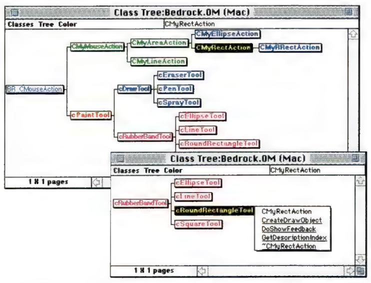
Object Master provides you with all the editing and navigational capabilities you require to write code productively. For example, the Class Tree window graphically displays your project's class hierarchy and allows you to expand and collapse "branches" and open multiple windows displaying specific code components.

Power Macintosh Support

ACI offers two versions of this outstanding programming tool: Object Master and Object Master UNIVERSAL. Both run on the traditional 68K-based Macintosh computer and the new Power Macintosh, taking full advantage of the features of each platform.

Object Master supports C and C++ and works seamlessly with Symantec's THINK Project Manager and Metrowerk's CodeWarrior. Separate versions are available for the 68K-based Macintosh and the new Power Macintosh computers.

Object Master UNIVERSAL supports C, C++, Pascal, Modula-2, and all major compilation systems. It can be installed on both the 68K-based Macintosh and the new Power Macintosh.



"...Object Master pays for itself in a week, even at suggested retail price." –Macworld Magazine

See Object Master for Yourself

Call (800) 384-0010, and we'll send you a free demo disk of Object Master—because seeing really is believing.

ACI

ACI US Inc. 20883 Stevens Creek Blvd., Cupertino, CA 95014
Tel. 1 408 252 4444. Fax 1 408 252 0831. AppleLink D4444

LS Object Pascal

Powerful ▪ Faster ▪ Native

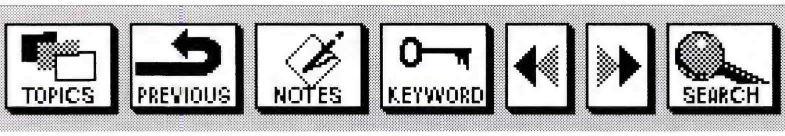
shipping on CD-ROM for PowerMac

now $399 was ~~$695~~

1-800-252-6479

Language Systems Corp. • 100 Carpenter Dr. • Sterling, VA 20164 • (703) 478-0181 • Fax: (703) 689-9593 • AppleLink: LANGSYS

EHelp



Add Multimedia Documentation To Your Applications And Now...

Build Stand-Alone Documentation With

EHelp 4.0

THE PREMIER DOCUMENT DISPLAY SYSTEM FOR THE MACINTOSH

- Integrated multimedia elements
- Flexible hot-spot linking
- Robust hypertext capabilities
- Sophisticated searching
- Easy integration with your applications
- Reads WinHelp™ and MSWord™ source
- Apple-event aware
- Function macros for inter-document jumps, control of user buttons, launching other applications, etc.
- World Ready for all Macintosh-supported languages
- Many other features

CALL
(800) 247-7890
OR
(919) 481-3517
FOR A
FREE EHELP
DEMO DISKETTE.

FSI
FOUNDATION SOLUTIONS INC

FAX (919) 481-3551

EHelp 4.0 is a trademark of Foundation Solutions, Inc.
EHelp 3.0 is still available and supported.

CMaster 2.0 installs inside of **THINK C 5/6/7** and **Symantec C++ For Macintosh**, and enhances the environment. No new editor to learn—CMaster builds on what Symantec has already provided. See for yourself why programmers from around the world use CMaster, and why MacUser gave CMaster 1 a **5-Mouse** rating.

CMaster 2.0 incorporates hundreds of features asked for by active programmers like yourself. These ads each hilite one special aspect CMaster 2.0 brings to the Symantec environment. EMail, FAX, or call for more information or a Demo Disk.

CMaster®

The Fast, Efficient Way to Edit THINK™ Code

Rick Johnston, Eartown Movies:
I ordered the upgrade the second I saw it—I don't think I could function without CMaster!

✈ Speed

Faster programming is CMaster's primary feature: for example, writing a new function. Use the Marker and GoBack menus to roam in the current file or project grabbing code snippets, storing them in clipboard stacks. Enter the function outline, then paste back the snippets as needed. Use expandable Glossaries and Macros to speed new code development. Press a key and move to the function start or the end of the automatics, and enter some (or use the "Enter Automatics" feature.) Hilite questionable variables, and CMaster searches for usage—if unused, CMaster intelligently hilites the declaration for quick deletion. Press ENTER to *snapback* to the previous edit location to finish up. Use CMaster to produce a formatted prototype, press again, and the matching ".h" file comes foremost, or CMaster finds and opens it for you. Voilà tout!

CMASTER PACKAGE ONLY $129.95
30 DAY MONEY-BACK GUARANTEE

Jersey Scientific, Inc 291 Springfield Av Suite 201, Berkeley Heights NJ 07922 TEL: 212.736.0406 FAX: 908.464.3458
APPLELINK: JERSCI • CIS: 70400,3361 • MASTERCARD, VISA, CHECKS, AND CORPORATE POs ACCEPTED.
$5 S&H US Post ($10 UPS BLUE, $19 Intl Express) • E-mail orders accepted— send VS/MC number, exp. date, and ship address
Free Demo Disk: Available on the Internet, CSI, and AppleLink. Or FAX, Email, or Call and ask for it.

THINK is a trademark of the Symantec Corporation. CMaster is a registered trademark of Jersey Scientific, Inc.

called CStream_myContents.cp. The contents of which should look something like the following:

```
#include "CStream.h"
#include "myContents.h"

#pragma template_access public

#pragma template PutObject(CStream&, myContents*)
#pragma template GetObject(CStream&, myContents*&)
#pragma template PutObjectReference(CStream&, myContents*)

#include "CStream.tem"
```

Q. *I am trying to create an AppleScript to build a project with no user interaction. I am having trouble with the 'save' script command. When I use the Script Editor to record the steps for building an application, the Script Editor generates the script command "save project document 'ABC Publish.x' as a project type". When running the script, I get an error: "'THINK Project Manager got an error: Can't make some data into the expected type." How can I edit the script so that it works correctly?*

A. The following script will do the trick:

```
tell application "THINK Project Manager"
    compile project document 1
    save project document "testtypes.1" to file
    "HD:Development:...etc...:Filename"
    as a "PRTP"
end tell
```

Q. *When I create a class which has a member function called SetItem(), I get a preprocessor error, "Three actual arguments expected for item." when trying to compile. Why doesn't this work?*

A. In the new universal headers, Apple changed the names of the GetItem and SetItem functions in <Menus.h> and added macros using the old names for backward compatibility. Unfortunately, these new macros will break all code that uses the identifiers GetItem and SetItem.

In order to fix this, in :Mac #includes:Universal Headers:<ConditionalMacros.h>, change #define OLDROUTINENAMES 1 to #define OLDROUTINENAMES 0 and then re-precompile <MacHeaders> and <MacHeaders++>. This will disable the new macros in <Menus.h> and in other headers. As a consequence, you will get "missing prototype" errors if your program uses the old function names.

Q. *When I compile a project, I get an error "Stack approaching limit. Use #pragma large stack." When I use* `#pragma large stack`*, I get a syntax error. Why?*

A. The directive `#pragma large stack` is not recognized by the current compiler. However, this message indicates that you are dangerously close to a code overflow. You should reduce the number of static variables in the offending source file. Alternatively, you can dynamically allocate your variables to reduce stack requirements.

dtF The Relational Database System

What is dtF...

dtF is a true relational database management system for C/C++ application development. dtF features SQL, full transaction control, error recovery and client-server architecture.

dtF is royalty free...

Create and distribute as many applications as you like, royalty free!

dtF is fast...

When it comes to performance dtF is in a class all its own. dtF utilizes a proprietary query optimization and caching scheme to obtain unparalleled performance.

dtF is efficient...

dtF was developed by Macintosh developers for Macintosh developers. Applications developed with dtF will be compact in both single and multi user form. Perfect RDBMS for Powerbook applications.

dtF is SQL...

dtF supports an efficient subset of ISO standard SQL that is optimized for speed and ease of use.

dtF is safe...

Full transaction control and error recovery guarantee maximum data protection even after sudden system crashes. dtF databases are compressed and encrypted to protect against all unauthorized access, even disk editors.

dtF is easy...

Integrated data dictionary, security, automatic index selection, query optimization, deadlock detection and error recovery allow you concentrate on your application.

dtF is true client server...

dtF client server architecture insures efficient use of network bandwidth and optimum data security. dtF will not bog down your network like systems that use file sharing.

dtF single user...

Client server applications will not be stranded on the LAN. Relink your application with dtF single user and you will have a no compromises high performance stand alone.

dtF is for you!

For maximum performance, realistic licensing policy and reasonable pricing you can not beat dtF. No static preprocessors, only a dynamic, fully featured SQL. The C/C++ API is identical and fully portable over all supported platforms in both single and multi-user environments. dtF is ideal for use with TCL and MacApp.

dtF Platforms and Compilers...

Available for Macintosh System 7.x and native Power PC. dtF Server requires a 68020 or better. MPW C/C++ and Symantec C/C++ 5.x and higher are both supported.

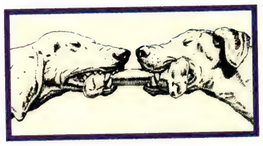
dtF Evaluation	$129
dtF Macintosh SDK	$695
dtF LAN Macintosh SDK*	$1595
dtF Server	$1295

dtF Americas, Inc.
12545 Olive Blvd, Suite 130
St. Louis, MO 63141
800.DTF.1790 Voice
314.530.1697 Fax
AppleLink: DTF.AMERICA

SLAB theta group

dtF is also available for DOS, Windows, OS/2 and several flavors of UNIX.

PROGRAMMER'S CHALLENGE



By Mike Scanlin, Mountain View, CA

RUBIK'S CUBE

There are several things I'm good at. Solving Rubik's Cube isn't one of them. One of my sadist friends gave me one last Christmas. I scramble it all the time and leave it on my desk at work. Then I watch in amazement as any one of several co-workers walks up to it and solves it within the amount of time it takes to ask "Mike, what's the fastest way to do buffered file I/O?" It's very frustrating.

This month's challenge idea comes not only from my own inadequacy but also from Jim Lloyd (Mountain View, CA). The goal is to solve Rubik's Cube.

The prototype of the function you write is:

```
int
SolveRubiksCube(cubePtr)
RubiksCube *cubePtr;
```

Since I am so pathetic at solving the cube, I do not have enough insight to design an effective data structure to represent it. You get to define the RubiksCube typedef anyway you like.

Each call to SolveRubiksCube should make exactly one move. A move means turning any side any direction by 90 degrees. The function returns one of 3 values on each call to it:

-1	illegal cube condition, can't be solved
0	made a move, but not done yet
1	made the last move, it's solved

Your routine will be called in a loop from my test bench something like this:

```
ScrambleCube(cubePtr);
do {
  x = SolveRubiksCube(cubePtr);
} while (x == 0);
```

Minimizing the number of moves to solve the cube is not important. Minimizing the total time required to solve the cube is important. If your SolveRubiksCube routine needs to keep track of some kind of state info or cached solution info between calls to it then it may use static variables to do so (or, you could incorporate that info into the RubiksCube data structure).

It is not required to detect an illegal cube condition on the first call to SolveRubiksCube, just so long as it is detected eventually (i.e. don't go into an infinite loop if someone removes a couple of squares from your cube and puts them back in such a way that it makes the cube unsolvable).

Because I will need to be able to watch your routine's progress, you will need to write two conversion routines:

```
void
MikeCubeToRubiksCube(mikePtr, rubikPtr)
MikeCube    *mikePtr;
RubiksCube *rubikPtr;

void
RubiksCubeToMikeCube(rubikPtr, mikePtr)
RubiksCube *rubikPtr;
MikeCube    *mikePtr;
```

The purpose is to convert your cube from whatever data structure you come up with into something my test bench can understand. It only knows about MikeCubes:

```
typedef struct CubeSide {
  char   littleSquare[3][3];
} CubeSide;

typedef struct MikeCube {
  CubeSide face[6];
} MikeCube;
```

A cube has 6 CubeSides. If you look at a cube straight on then the indexes are: 0 = top, 1 = left side, 2 = front, 3 = right side, 4 = bottom, 5 = back. Within each CubeSide there is a 3x3 array of littleSquares (first index is row, second index is column). Each littleSquare is a number from 0 to 5, where each number represents one of the 6 colors that make up a cube.

So, a solved cube would have all 9 of the littleSquares for each side set to the same value. A solved cube that had the rightmost vertical column rotated counterclockwise 90 degrees would look like this:

```
   005
   005
   005
111220333
111220333
111220333
   442
   442
   442
   554
   554
   554
```

The top part (the 005 lines) represent the top of the cube after the rotation. The 220 pieces are what you see on the front, the 442 lines are what you see on the bottom and the 554 pieces are what's on the back. The value of `mikePtr->face[0].littleSquare[1][2]` is 5. The value of `mikePtr->face[0].littleSquare[1][1]` is 0 (the center square of the top).

Write to me if you have questions on this. Your conversion routines are not going to be part of your times so it doesn't matter if they're slow. Have fun.

Two Months Ago Winner

Based on the number of entries I received for the How Long Will It Take Challenge I would have to say that software time estimates are even more difficult than their well-deserved reputation implies. Either that or the typical Programmer Challenge entrant isn't concerned with schedules (or, at least, this Challenge). In any case, congrats (and thanks) to **Jeremy Vineyard** (Lawrence, KS) for winning. I suppose the fact that he was the only person to enter helped his entry somewhat but there are signs of some careful thinking in his entry as well. (Although I must confess that I find the inclusion of a Jolt Cola parameter completely useless. Mountain Dew on the other hand...) This is Jeremy's second 1st place showing (he also won the Insane Anglo Warlord challenge in February, 1993). Nice job!

Jeremy's entry refers to a book by E. Barne called "Estimation of Time-Space Development Theorems" which I wasn't, unfortunately, able to track down before my column deadline. Sounds like something to check out, though, if you're involved with scheduling.

Here's Jeremy's winning solution:

Top 5 Excuses Why This Project Was Late:

1. "I though you said it was due NEXT month!"
2. The Product Manager quit after we missed the beta acceptance deadline.
3. Somebody activated the fire alarm while we were holding a brainstorming session, and all our ideas were lost.
4. QA didn't keep us up to date on active bugs.
5. I just couldn't wait to try out the new game of solitaire that came with System 7.5!

The skill of the product manager is the key to a successful product.

```
enum {
   // Manager ratings.
   noManager = 0,
   lazyManager = 1,
   poorManager = 2,
   okManager = 3,
   motivatedManager = 4,
   excellentManager = 5
};

enum {
   // System versions.
   system7 = 7,
   system6 = 6,
   system5 = 5,
   system4 = 4,
   system3 = 3,
   system2 = 2,
   system1 = 1
};


// It assumed that the better the manager is, the more lines
// the engineers will be motivated to write.
#define juniorLines       (100 + (20 * manager))
#define seniorLines       (200 + (25 * manager))

// Every large project will have an overhead for planning,
// designing, and preparing for development.
#define overhead          (linesC / 5000)

// These ratios are based on Barne's matrix equations
// related to the study of time-space software development
// theorems. (trust me!)
#define PPCNativeRatio     (float) (1.75 - (.02 * powerMacs))
#define joltRatio          (float) (.7)
#define systemSupportRatio    (float) (7.0 / systemSupport)

// Here again,the manager can motivate both QA and UI
// people to be more productive.
#define qaRatio    1 - (.1 + (.01 * manager) * qaPeople)
#define uiRatio         (float) ((1.2 - \
       (.01 * manager)) * (seniorEng - uiPeople))

// Localization- a nightmare. Enough said.
#define localizedRatio     (float) (1.35 * localized)
```

SoftwareTimeEstimate

A simplified algorithm that accurately calculates the time in calendar days for any Macintosh software product to be completed.

The complete algorithm and collection of equations can be found in E. Barne's *Estimation of Time-Space Development Theorems*, published by Academic Press.

Implementation by Jeremy Vineyard, Lawrence, KS

SoftwareTimeEstimate

```
unsigned short SoftwareTimeEstimate(linesC, seniorEng,
  juniorEng, systemSupport, PPCNative, powerMacs, manager,
  jolt, qaPeople, uiPeople, localized)

  // Estimated # lines of source code in C.
  unsigned long linesC;

  // # of competent engineers with more than 5
  // years experience.
  unsigned short seniorEng;

  // # of junior engineers with less than 2
  // years experience.
  unsigned short juniorEng;

  // Earliest version of system software supported by
  // product (1-7)
  unsigned short systemSupport;

  // TRUE if product will be PowerPC native.
  Boolean PPCNative;

  // # of PowerMacs available to developers (1-30).
  unsigned short powerMacs;

  // Rating of manager 1-5 (1 is poor, 5 is excellent)
  // or 0 if none.
  unsigned short manager;

  // TRUE if company has steady supply of Jolt cola.
  Boolean jolt;

  // # of trained, in-house testers (most important).
  unsigned short qaPeople;

  // # of people responsible for making decisions about UI.
  unsigned short uiPeople;

  // # of Languages product must be localized to before
  // shipping or 0 if none.
  unsigned short localized;
{
  float      time, ratio;
  short      linesADay;

  // Calculate the lines of source code that can be
  // produced per day based upon the number of engineers
  // working on the product.
  // (Must have at least one engineer)
  linesADay = (seniorEng * seniorLines)
              + (juniorEng * juniorLines);

  // Estimate the # of days it will take to produce the
  // specified amount of code.
  time = (linesC / linesADay) + overhead;

  // The farther backwards this product is compatible with
  // system versions, the more time it will take to develop.
  // systemSupport should be from 1 (System 1.0) to 7.
  time *= systemSupportRatio;

  // More development & QA time will need to be spent on
  // a product that is PowerPC native. The more Power Macs
  // that are available to developers, the faster the
  // development & testing process will proceed.
  if (PPCNative)
    time *= PPCNativeRatio;

  // o JOLT COLA! o
  // o GIve the PRogramMeRS that EXtra ZING! to geT thEm o
  // o MOTIVATED!! o
  if (jolt)
    time *= joltRatio;

  // Quality assurance is one of the most important
  // aspects of a product's construction. Without it, the
  // product will fail to reach the user base because of
  // too many bugs. QA can never be overstaffed!
  time *= qaRatio;

  // User interface has a limited potential because people
  // are likely to have very set opinions, and if there
  // are too many UI people, it can kill a product. There
  // should never be more UI personnel than engineers.
  if (uiPeople > seniorEng)
    time *= uiRatio;

  // The more languages the product must be localized to
  // before it can be shipped has a drastic effect on the
  // development time.
  if (localized)
    time *= localizedRatio;

  return (short) time;
}
```

All TEXT is not created equal.

Is your application taking *cut & paste* a little too literally when it creates text? Are you tired of living in a monotype world? **PAIGE**™, our text & page layout programming library, is the ultimate cross-platform solution.

PAIGE provides the most sophisticated features/functions in the business. These include:

- ***Stylized Text***
- ***Shapes & Containers***
- ***Text Wrapping***
- ***Embedded Objects***
- ***Hypertext Links***
- ***Virtual Memory***
- ***Style Sheet Support***
- ***Multi-Level "Undo"***
- ***Royalty Free***

So why should you join the **PAIGE** revolution? TIME. Most programmers don't have time to reinvent the wheel.

PAIGE was designed using no global variables and machine specific code has been isolated into two small source files. This strategy allows you to move your application to other operating systems or platforms by changing only platform specific code while maintaining full data and application compatibility.

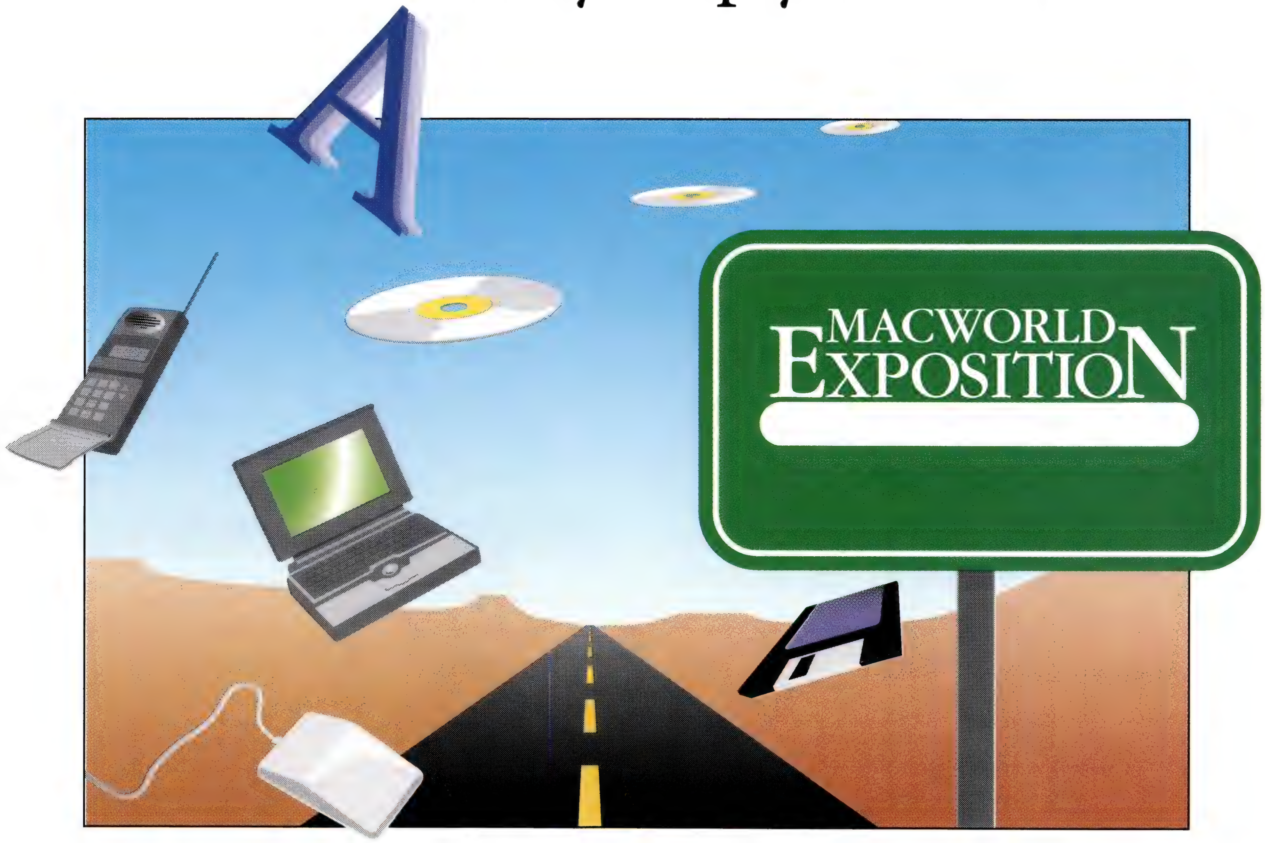
Join the hundreds of major software publishers using **PAIGE** as their total text solution. For complete technical/pricing summary contact **DataPak Software** at 800-327-6703 or 206-573-9155.

Macintosh • Power Macintosh • Windows

To receive information on any products advertised in this issue, send your request via Internet: productinfo@xplain.com

Forget Gas, Food & Lodging

On the Information Superhighway this is the only stop you'll need.



Don't want to be bypassed on the Information Superhighway? Then plan a detour to MACWORLD Expo. Here you'll test drive the products and services that enable you to maximize the potential of the Macintosh now and down the road.

MACWORLD Expo is your chance to see hundreds of companies presenting the latest in turbo-charged Macintosh technology. Make side by side comparisons of thousands of Macintosh products. Learn from the experts how to fine-tune your system and what products will keep your engine running smooth. Attend a variety of information-packed conference programs that provide the skills and knowledge to put you in the driver's seat. So pull on in and take that new Mac for a spin.

Add a stop at any of our upcoming MACWORLD Expo events to your information roadmap. With shows in **San Francisco, Boston** and **Toronto,** we're just around the next bend.

Please send me more information on MACWORLD Expo.
I am interested in: ❑ Exhibiting ❑ Attending

❑ San Francisco ❑ Boston ❑ Toronto

Name ______________________

Title ______________________

Company ______________________

Address ______________________

City/State/Zip ______________________

Phone ______________ Fax ______________

Mail to: Mitch Hall Associates, 260 Milton Street, Dedham, MA 02026
Or Fax to: 617-361-3389 Phone: 617-361-8000

MACTECH

By Steve Sisak, Articulate Systems

Adding Threads To Sprocket

Programming like you had a "real" OS

INTRODUCTION

Last month, in Randy Thelan's article "Threading Your Apps", you learned a bit about the theory behind the Threads Manager and how to call its routines. This month, we'll skip past the theory and focus on how you can combine the Threads Manager and the AppleEvent Manager, with the help of a couple of small libraries, to make the job of writing a modern Macintosh application much easier. Rather than cover every aspect of the Threads Manager, we will concentrate on a few key features that can really save you time. In fact, if you're using the libraries as-is, you may never need to call the Threads Manager directly at all. The libraries, AEThreads and Futures, are useful as both stand-alone libraries and as non-trivial examples of multi-threaded code. In addition, in what I hope continues as a good trend, I have integrated them into the MacTech application framework, Sprocket. We'll start with a quick overview of the libraries, continue with a description of how to use them, and finally delve into the details of how they're implemented. Even if you only plan to use the libraries as-is, this section should help you to understand threaded programming and may also help you avoid traps and pitfalls I've already encountered.

The application model we'll be using is a threaded extension of the fully factored application that Apple has been begging us to implement for the last few years – it is split into a user interface portion and a server portion which communicate with each other using AppleEvents. The difference is that, instead of handling each event sequentially, the AEThreads library allows the server portion of the application to handle multiple events concurrently and asynchronously, each in a separate thread of execution. It takes care of all of the bookkeeping involved with spawning and cleaning up after the threads, suspending and resuming the AppleEvents, and reporting errors to the AppleEvent Manager. Further, the Futures library provides routines that allow you to write client code in a traditional linear style, without idle procs, and to ask multiple questions of multiple servers simultaneously. And, even better, the Threads and AppleEvent managers do all of the hard work for you!

I first got interested in threaded AppleEvent code while working on a credit card validation server (*Credit Now!*, Appropriate Solutions). The intent was to develop a small application which could receive information about a purchase, dial up the credit card network, process the transaction and return an authorization code. Sending an AppleEvent seemed like the logical thing to do. The problem was that, with the phone call involved, the event took approximately 45 seconds to generate a reply. Further, during that time, we had to service the serial port, run a communication protocol which was best expressed as a loop, and service the user. (After all, no user would tolerate their machine locking up for 45 seconds.) This

Steve Sisak – Steve lives in Cambridge, MA, with three Macintoshes, two Newtons, and two neurotic cats. He referees Lacrosse games, plays hockey, drinks beer, and searches for the perfect vindaloo. He also finds time to work on a speech recognition system for Articulate Systems during the day, a multithreaded communication server and thread-savvy studio library at night, and a variety of odd jobs the rest of the time.

sounded like a good job for the then just-released Threads Manager. I'm not sure where I got the idea of spawning threads using AppleEvents, but I'm pretty sure it involved beer. Anyway, I've used the code in several products since and have been very happy with it.

A word about the code. One of my pet peeves with example code has been that handling errors is left as an exercise for the programmer. This is not the case with AEThreads and Futures. I've made every attempt to ensure that the libraries meet commercial quality code standards. While I must make all the usual disclaimers, I use this code in shipping products, so I'll fix any bugs you report. I have provided the full source to offer you some insight into some nuances of threaded programming and to allow you advanced programmers out there to fine tune them. (I use slightly different versions to support TCL, MacApp, and PowerPlant exception handling.) Also, until Sprocket has a true Thread class (coming soon), you may want to edit the SwitchIn and SwitchOut routines to save additional context unique to your application.

You may also notice that this article makes no mention whatsoever (except in this paragraph) of preemptive threads, semaphores, critical sections or a number of other things you may have heard of in your operating systems class. This is intentional. This article and code are intended to save you time. Preemptive threads are not supported on the PowerPC. Having them in your system means you have to think about things like semaphores, resource contention, deadly embrace, etc. This does not save you time. Enough said.

So, without further ado, let's get to the meat of the article.

AEThreads

The first thing you need to do to implement a multi-threaded server is figure out how to spawn a new thread. If you've been reading the Threads Manager documentation and Randy's article last month, you know that to start a new thread, all you have to do is call NewThread(), right? How do you get parameters to it? How do you get results back? You dutifully pack up all the information into a big structure in a local variable and pass its address as the threadParam to NewThread. You pass the address of another variable as the threadResult parameter. But when you run your code, you discover, after many trips to Macsbug, that the routine that called NewThread() has long since returned by the time the thread actually runs (leaving threadParam and threadResult pointing into hyperspace). There's got to be an easier way to get information to a thread.

Well, how about just sending it an AppleEvent? OK, a thread isn't an application, so you can't send an event to it directly, but using the AEThreads library you can send an event to yourself and have it handled by a thread. In fact, AEThreads has only one public routine – AEInstallThreadedEventHandler(). To use it, just call it where you would have called AEInstallEventHandler() to install your handler for the event. AEInstallThreadedEventHandler() takes the same parameters as AEInstallEventHandler (except isSysHandler, since we need our applications's A5 world) plus the ThreadOptions and stack size parameters from NewThread. When an event comes in, AEThreads automatically suspends the event, creates a new thread and calls your handler from within it. When your handler returns, it cleans up, resumes the event, and returns the reply to the sender. Notice that if the Threads Manager is not present and you don't do anything that requires multiple events to be handled simultaneously, you can just call AEInstallEventHandler() instead of AEInstallThreadedEventHandler() and your application will still run, except that incoming events will be handled synchronously.

Yielding

Now that we have all of our AppleEvents handled in separate threads – we're done, right? Well, not quite. Recall that we're running in a cooperative multitasking environment. This means that it is our responsibility to give CPU time to other threads; if we don't, then our thread will finish synchronously, completely missing the advantage of using a thread to handle the event.

The Threads manager provides a number of routines for scheduling thread execution. The simplest of these is YieldToAnyThread(), which you can call at any time you think would be good to give someone else some CPU time. The important thing to note is that, unlike WaitNextEvent(), YieldToAnyThread() is a very inexpensive call, so you shouldn't be afraid to call it often. (You should, however, have a thread-aware event loop, like the one Dave Falkenburg put into Sprocket, to make sure your main thread doesn't call WaitNextEvent too often).

Blocking I/O

Another technique for yielding CPU time involves blocking I/O routines. These are routines which look like high level synchronous I/O routines, except that they know enough to block your thread and give up the CPU while the I/O is executing. For instance, a blocking version of FSRead() might look like this:

```
pascal OSErr ThreadFSRead(short refNum,long *count,void
*buffPtr, long timeout)
{
  ParamBlockRec pb;

  pb.ioParam.ioCompletion = nil;
  pb.ioParam.ioRefNum = refNum;
  pb.ioParam.ioBuffer = buffPtr;
  pb.ioParam.ioReqCount = *count;
  pb.ioParam.ioPosMode = 0;
  pb.ioParam.ioPosOffset = 0;

  timeout += TickCount();

  PBReadAsync(&pb);

  while (pb.ioParam.ioResult > 0 && TickCount() <= timeout)
  {
    YieldToAnyThread();
  }
```

New BBEdit 3.0 High Performance Text Editing

"It still doesn't suck."

—Tom Emerson, Software Engineer, Symantec Corporation

Accelerated for Power Macintosh

BBEdit on a PowerMac gives you even faster text editing, searching and transformation.

More than just a pretty face, BBEdit 3.0 kicks ass. Among other things, it offers:

- Scripting from AppleScript, Frontier 3.0, or any other OSA-compatible scripting system.
- Macintosh Drag and Drop to speed text editing.
- Integrated PopupFuncs™ technology for easy navigation of source files.
- Integrated support for Symantec C++, and THINK Reference, and MPW ToolServer.
- The ability to compare projects *or* folders, as well as files.
- A non-stick coating for easy clean up.

...and the list goes on...

Now we're screaming.

BBEdit edits, searches, transforms, compares and massages text between two and ten times faster on a Power Mac 8100 than on a Quadra 610.

"...most people will find that BBEdit offers everything they need and is easy to use."

— MacTech, June 1994

Tell me more! Tell me more!

Check the Mail Order Store section at the back of this issue for info about ordering BBEdit.

For a free demo disk, send us e-mail:

Internet: bbedit@world.std.com
AppleLink: BARE.BONES CompuServe: 73051,3255

We also have "It doesn't suck." t-shirts that don't either. Call us to order. © 1994 Bare Bones Software, 1 Larkspur Way #4, Natick, MA 01760. Phone: 508/651-3561. Fax: 508/651-7584.
By the way: BBEdit is a trademark of Bare Bones Software, Inc. All other trademarks are properties of their respective holders.

```
  *count = pb.ioParam.ioActCount;
  return pb.ioParam.ioResult;
}
```

Now, replacing calls to FSRead() with ThreadFSRead() will allow other threads to execute while the I/O is in progress. We have also added a timeout mechanism which is useful for serial I/O. Note that this technique requires very little modification to existing code. Further, Apple promises to add blocking I/O to the Mac OS in the future. Writing your code in this way prepares you for the future. You may also notice that we didn't use any of the fancy Thread Manager routines to suspend the thread, etc. This is because it is possible for the asynchronous I/O call to complete while we were trying to put the thread to sleep, leaving it stuck that way. While there are ways around this, it's not worth the hassle. YieldToAnyThread() is cheap, and this technique gives a robust timeout mechanism for free. (If you didn't understand that, don't worry, you don't need to).

It is important to note that, depending on the driver you're calling, the call to PBReadAsync() may complete synchronously, resulting in no yield. This is true for floppy and pre-SCSI Manager 4.3 hard disk reads. In these cases you may want a more sophisticated routine which reads blocks, say 4k, yielding between each block. Also, in the case of the serial port, you may want to issue a control call first to see how many bytes are actually waiting. But these are topics for another article...

Futures

One of the biggest hassles of writing AppleEvent-based client-server code (or any communications code in the Macintosh) is that you must rewrite any time-consuming algorithm into a state machine built around idle procs. This is because you must constantly service the event loop by calling WaitNextEvent(). This is tedious, time consuming, error prone, and generally not fun. If you've got some spare time you just really want to waste, try implementing the ZModem protocol this way.

Enter the Threads Manager. Now you can just spawn a thread, implement your algorithm as a series of nested loops (just like a "real" operating system), and sprinkle a few calls to YieldToAnyThread() through your code. This works great if you're talking to a serial port, but what if you want to send AppleEvents to another application (or yourself) in the loop. Then you're back to writing state machines. (Or some very complicated Apple Event idle procs)

In his article "Threaded Communication with Futures" in **develop** issue 7, Michael Gough described a particularly elegant solution to this problem, called the Futures Package, which provided a blocking I/O style interface to the AppleEvent manager based on the Threads Manager's predecessor, the Threads Package. In fact, it goes even further by performing a kind of delayed evaluation which allows the client code to send events to multiple servers without waiting until it needs to use a reply which hasn't returned yet. Unfortunately, support for the Futures Package was discontinued when the Threads Package was replaced by the Threads Manager. Apple even pulled the example code off the **develop** CD.

Last December I was talking to Brad Post and Eric Anderson about the Threads Manager at a PowerPC training session and asked what ever became of the Futures Package. I was told that, while it was really cool, nobody had gotten the time to update it and it didn't look like it was going to happen any time soon. I begged. The next day Brad gave me some source that hadn't been compiled yet to see if I could get it working. I promised to try. After a couple of late nights and some serious hair pulling trying to find a missing ThreadEndCritical(), (I later removed all the critical section code because the AppleEvent manager can't be called at interrupt time and nobody else has any business looking at the data anyway) I got the package working. I give this long-winded explanation because the small amount of code you see here is the result of a lot of people's work and I'm not entirely sure who gets credit for what, but the end result is extremely cool, elegant, and useful.

The key function in the Futures package is Ask(). Ask() functions like AESend(), except that it returns immediately. The trick is that its reply AppleEvent is actually a future (or thunk to you Scheme programmers, and to resolve Randy's dangling pun from last month). As long as you don't attempt to extract any data from it, your code continues to execute normally. However, if you try to extract information (using AEGetParamPtr() or any of its friends) before the reply has actually returned, the Futures library will suspend execution of your thread until the reply actually returns. The result is that, to your thread, it looks like AEGetParamPtr() returned immediately too, but in fact the CPU was transparently given to other tasks until your request could be fulfilled. **Important:** If you want to be able to process incoming events during a call to Ask(), make sure you're not calling it from the main thread. This is what AEThreads is designed to help you do – just install your server routine as a threaded handler.

Disclaimer

You may discover, after reading the code, that the functionality of the Futures library really belongs in the AppleEvent manager. It does. In fact, I believe that adding kAEFutureReply as a send mode in AESend() would take care of it at the API level. (Is anyone at Apple listening?) The source code is provided here because I cannot guarantee that it is entirely bug free. However, you should be very cautious in modifying it. It uses a pair of special handlers in the AppleEvent manager which were added to support the original futures package; they're not documented for third-party use, and I strongly urge that you not use them for anything else. There is also a design limitation if the reply to your event is never received (because of a lost network connection or some other reason) you will not get a timeout error and your thread will stay suspended indefinitely. I plan to fix this in the future, and it may not be a big problem in many cases, but please be aware of it. If anyone out there wants to add this functionality, I will gladly include it in a future release.

```
AEThreads Reference

AEInstallThreadedEventHandler

pascal OSErr AEInstallThreadedEventHandler(
            AEEventClass theAEEventClass,
            AEEventID theAEEventID,
            AEEventHandlerProcPtr handler,
            long handlerRefcon,
            ThreadOptions options,
            Size stacksize);
```

AEInstallThreadedEventHandler is used to install a handler for an AppleEvent which is to be handled in a thread. The first four parameters are the same as for AEInstallEventHandler(), the last two are passed to NewThread() to create a thread for your handler. You may use any of the parameters documented in the threads manager except kNewSuspend (which would cause your handler to never be called).

Futures Reference

For a full description of the futures library, see Michael Gough's **develop** article. A quick summary is provided here for your convenience.

InitFutures

```
pascal OSErr InitFutures(void);
```

Call InitFutures() once at the beginning of your program to initialize the futures package.

Ask

```
pascal OSErr Ask(AppleEvent* question, AppleEvent* answer);
```

Call Ask() instead of AESend() to send an AppleEvent. The trick to efficient code is to call Ask() as early as possible and look at the result as late as possible in order to give the server the maximum amount to time to process your request.

IsFuture

```
pascal Boolean IsFuture(AppleEvent* message);
```

Call IsFuture() to test if the reply from a call to Ask() is a future without stopping your thread if it is. If IsFuture() returns true, the reply hasn't returned yet and you should probably go work on something else.

A DEVELOPER'S BEST FRIEND

Don't Bury Your Profits!

Software producers will lose over 7 billion dollars worldwide to software piracy this year. This number would be even higher if it weren't for PACE Anti-Piracy. For nearly a decade, PACE has provided Macintosh software producers with the most effective means of protecting against piracy.

Your Sales Will Increase But Your Tech Support Calls Won't.

Not just copy protection, **MacEncrypt**™ is a complete key diskette and verbal authorization system. With **MacEncrypt** you can protect your profits while providing your customers with the flexibility and ease of use they require. Unlike other products on the market, especially hardware keys and decoder rings, your customers will have a positive experience with your **MacEncrypt** protected software.

Features:

- Protects all disk formats, including CD ROM's. Requires no special media.
- Authorize your customers over the phone.
- Protected software can be installed to hard disks, compatible with disk optimizers.
- Protected diskettes can't be copied by disk copy programs.
- Turn-key installation for applications; you don't have to be a programmer to protect your app.
- The protection can be custom installed to any software–not just apps.
- Complete integration with the Apple Installer and StuffIt InstallerMaker™.
- Automatically protects against virus infection.

For a protected sample diskette and more information about the MacEncrypt system, call or write
PACE Anti-Piracy **1082 Glen Echo Ave, San Jose, CA 95125**
(408) 297-7444 • Fax (408) 297-7441 • AppleLink PACE.AP

PACE
ANTI-PIRACY

"Copy protection that isn't superb costs a company more than it saves. PACE products offer unparalleled security and compatibility at a great price. PACE is a rare company dedicated to making the unthinkable possible: copy protection that's hassle-free, for both my end-users and my support staff."

Stephen Greenfield
President
Screenplay Systems, Inc.

"PACE Anti-Piracy software is a reliable means of limiting illegal distribution of our software and enforcing our licensing agreements."

Mike Aaron
Opcode Systems

BlockUntilReal

pascal OSErr BlockUntilReal(AppleEvent* message);

Call BlockUntilReal() to force the current thread to block until the reply actually comes back.

How AEThreads Works

The AEThreads library is implemented as a chain of three main routines (AEInstallThreadedEventHandler(), SpawnAEThread() and AEThread()) and a pair of custom thread switchers.

AEInstallThreadedEventHandler() is a fairly simple routine. Basically, it allocates a pointer in the application heap, copies its parameters there and calls AEInstallEventHandler() for the event specified with SpawnAEThread() as the handler and the pointer as the refcon.

When an event comes in, SpawnAEThread() is invoked. Its responsibility is to get the new thread started. First, it copies the parameters needed by the new thread into a local structure and then calls NewThread() with AEThread() as the threadProc and the address of the parameter structure as the threadParam. Next it enters a yield loop, yielding the thread it just created and watching a flag in the parameter block for a signal that the thread has started. This is necessary because, although the thread has been created, the threadProc hasn't been called yet and the thread parameters are in a local variable. If we returned to the AppleEvent manager now, the threadParam would be a dangling pointer when the thread actually ran. This can also be avoided by allocating the threadParam as a pointer or handle. I have chosen not to do this here, however, because I don't want to fragment the heap more than necessary and the semaphore structure allows us to get an error message from the threadProc if it has difficulty getting itself up and running. (OK, I mentioned semaphore again, but at least you didn't have to deal with it).

AEThread() is the wrapper routine which actually calls your handler. The first thing it does is copy its parameters into local variables so they'll still be around when SpawnAEThread() returns. Next it installs a pair custom switcher which saves and restore any global context unique to used to our thread. Metrowerks C++ stores a linked list of stack-based object whose destructors must be called in compiler-generated global variable `__local_destructor_chain`. You must save and restore it here or you will be very sorry. Also if you are using a class library with exception handling, you would save and restore the failure stack here (gTopHandler for MacApp;

Developers:

PatchWorks™

Builds Updaters Without Programming

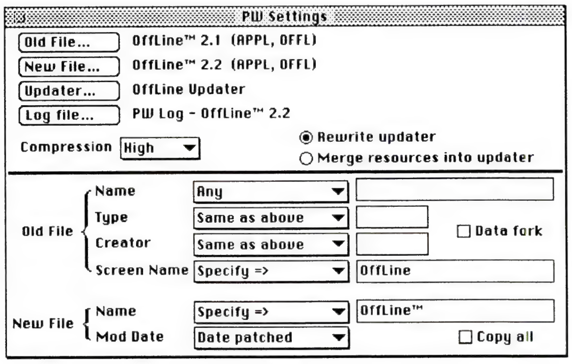
PatchWorks has many options, but only one function: to create updater applications for distribution to end users over channels that may be non-secure (e.g., BBSs).

Before the advent of ***PatchWorks***, creating an updater was a project in itself, one that consumed valuable programmer time which could more profitably be spent on revenue-producing projects.

With ***PatchWorks***, you create an updater in minutes. Since there's no coding or scripting, no bugs are introduced. Just fill in a dialog, and ***PatchWorks*** does the rest!

Distribute updaters frequently to reflect maintenance releases, and watch your tech support and fulfillment costs fall dramatically.

Most important, your customers will know you care.


PW Settings

Old File... OffLine™ 2.1 (APPL, OFFL)
New File... OffLine™ 2.2 (APPL, OFFL)
Updater... OffLine Updater
Log file... PW Log - OffLine™ 2.2
Compression: High
[✓ Rewrite updater] Merge resources into updater

Old File
- Name: Any
- Type: Same as above
- Creator: Same as above
- Screen Name: Specify => OffLine
- ☐ Data fork

New File
- Name: Specify => OffLine™
- Mod Date: Date patched
- ☐ Copy all

Features

- Works with apps, INITs, cdevs, fonts, drivers, etc.
- Updaters support up to 8 old versions
- Resource compression (diffing) produces small updaters
- Resource encryption makes updaters hacker-resistant
- Preserves personalization data (name, serial #, etc.)
- Updater distribution is **unrestricted & royalty-free**

Pricing: Begins at $195. Call for more information.

SNA

SNA, Inc.
2200 NW Corporate Blvd.
Boca Raton, FL 33431
Tel (407) 241-0308 • FAX (407) 241-3195

gTopHandler, gLastError, and gLastMessage for TCL 1.x). You may also want to save thePort, etc. This is necessary so that if a failure occurs in the thread, no handlers from outside the thread will be called. Now we suspend the event, clear the semaphore, and enter a yield loop so SpawnAEThread() can return to the AppleEvent manager.

The next time we get time, we are on our own. We now call the user's handler, wrapped in a failure handler, if available. If an error is reported here, we must manually pack the error code into the reply AppleEvent before returning. Next we resume the event and return to the Apple Event Manager. This will let the thread die and cause the reply event to be returned to the sender.

OK, it's a little confusing, but just remember that it was done here so you won't have to...

How Futures Works

The futures code is based around two undocumented (until now) callbacks in the AppleEvent manager which are accessed via AEInstallSpecialHandler(). One ('blck') [Randy, can we get these puppies some real names] is called whenever there is an attempt to read data from a reply event which hasn't arrived yet. The other 'unbk' is called whenever a reply arrives. We also use the refcon ('refc') attribute of the event to store a handle to a list of the threads currently waiting on a reply for this event.

InitFutures() simply installs DoThreadBlock() and DoThreadUnBlock() as the 'blck' and 'unbk' handlers, respectively.

If someone tries to extract data from a reply which is really a future, DoThreadBlock() is called by the AppleEvent manager. DoThreadBlock() then checks the 'refc' attribute of the event for a list of blocked threads. If it finds one, it adds the current thread to it; otherwise it creates a new list containing the current thread and puts it back in the reply. In either case it then suspends the thread which caused it to be called.

Whenever an incoming reply is detected, DoThreadUnBlock() is called for the event. If the refcon of the event contains a blocked thread list, DoThreadUnBlock() makes all of the threads in the list eligible for execution.

Ask() is simply a wrapper which calls AESend() synchronously with a timeout of 0 ticks, causing it to time out immediately and suppresses the timeout error. The reply will eventually come in, however, and be properly reported.

IsFuture() temporarily replaces DoThreadBlock() with a handler which does nothing and tests for a errAEReplyNotArrived error.

BlockUntilReal() just tries to extract a nonexistent parameter, 'gag!', from the event, forcing a block if the reply isn't real.

The Sample Code

The demo works as follows: Selecting "Ping" from the Debug menu sends an AppleEvent back to the application which calls the HandlePing() function in a thread. It brings up a PPCBrowser window asking you to select this or another copy of Sprocket as a target. It then enters a loop sending ping events to the target. For each ping, the target extracts the direct object from the event, stuffs in a reply and beeps to let you know it's doing its job. Selecting Ping2 does the same thing, but lets you choose two separate targets.

To see how this all works, take a look at the latest version of Sprocket and check out the files AEThreads.h and .c, Futures.h and .c, and FuturesDemo.h and.c. FuturesDemo.h is excerpted from Michael's **develop** sample code, updated to use AEThreads. There are also a few lines added to App.cp to set up the libraries and add menu handlers.

In Conclusion...

Well, that's about it. I hope both this article and the code are useful to you. In the future, I would like to continue to add technology to Sprocket. The next logical candidates are a simple Thread class, and an exception library (the other thing I can't live without). It's 3am and there's too much blood in my caffeine system, so good night...

DEVELOPER
DU
UNIVERSITY

Power Up with PowerPC !

Apple's Developer University has the right courses to jump start your PowerPC development efforts:

- ❑ **Programmer's Introduction to RISC and PowerPC self-paced training**
- ❑ **PowerPC BootCamp Class**

And, with over 19 courses, we offer the fastest way to get up to speed on the Macintosh, whether you're just getting started or seeking to understand Apple's newest development technologies.

For more information, contact the Apple Developer University Registrar by telephone at (408) 974-4897 or fax (408) 974-0544.

Developer University, Apple Computer, Inc. 1 Infinite Loop, MS 305-1TU, Cupertino, CA 95014

AEThreads.h

```
// AEThreads.h
//
// Copyright © 1993-94 Steve Sisak
//
// License is granted to use, modify, make derivative works, and
// duplicate this code at will, so long as this notice remains intact.
//

#ifndef __AETHREADS__
#include "AEThreads.h"
#endif
#ifndef __ERRORS__
#include <Errors.h>
#endif
#ifndef __APPLEEVENTS__
#include <AppleEvents.h>
#endif
#if __MWERKS__
#if defined(powerc) || defined(__powerc)
#include <CPlusLibPPC.h>
#else
#include <stdlib.h>
#include <CPlusLib68k.h>
#endif
#endif

typedef struct AEThreadDesc   AEThreadDesc;
typedef struct AEThreadParam  AEThreadParam;
typedef struct AESwapData     AESwapData;

struct AEThreadDesc              // Kept in the OS refcon
{
  AEEventHandlerUPP handler;     // The real handler
  long        refcon;            // The real refcon
  Size        stackSize;         // Stack size for handling event
  ThreadOptions   options;       // Thread options for event
  ThreadID      holder;          // used as a semaphore
};

struct AEThreadParam             // Used in spawning
{
  const AppleEvent* event;
  AppleEvent*     reply;
  AEThreadDesc*   desc;
  ThreadID      thread;
  OSErr         result;
};

struct AESwapData
{
  void*         fTopHandler;   // Top failure handler

#ifdef __MWERKS__
  DestructorChain*  fStaticChain; //
#endif
};

pascal void   SwitchInHandler(ThreadID threadBeingSwitched,
                          void *switchProcParam);
pascal void   SwitchOutHandler(ThreadID threadBeingSwitched,
                          void *switchProcParam);
pascal OSErr  SpawnAEThread( const AppleEvent *theAppleEvent,
                          AppleEvent *reply,
                          long handlerRefcon);
pascal long   AEThread(AEThreadParam* parms);

AEEventHandlerUPP gSpawnAEThreadUPP = nil;


#pragma segment foobar
                                 AEInstallThreadedEventHandler

pascal OSErr AEInstallThreadedEventHandler(
  AEEventClass    theAEEventClass,
  AEEventID       theAEEventID,
  AEEventHandlerUPP proc,
  long          handlerRefcon,
  ThreadOptions   options,
  Size          stacksize)
{
  AEThreadDesc* desc = (AEThreadDesc*)
                          NewPtr(sizeof(AEThreadDesc));
  OSErr      err   = MemError();

  if (gSpawnAEThreadUPP == nil)
  {
    gSpawnAEThreadUPP = NewAEEventHandlerProc(SpawnAEThread);
  }

  if (err == noErr)
  {
    desc->handler= proc;
    desc->refcon  = handlerRefcon;
    desc->stackSize = stacksize;
    desc->options= options;
    desc->holder  = kNoThreadID;

    err = AEInstallEventHandler(theAEEventClass,
                theAEEventID, gSpawnAEThreadUPP,
                (long) desc, false);
  }

  return err;
}

#pragma segment AEThread
```

SwitchInHandler

```
pascal void SwitchInHandler(ThreadID threadBeingSwitched,
void *switchProcParam)
{
  AESwapData* swap = (AESwapData*) switchProcParam;

#ifdef __MWERKS__
  swap->fStaticChain = __local_destructor_chain;
#endif
}
```

SwitchOutHandler

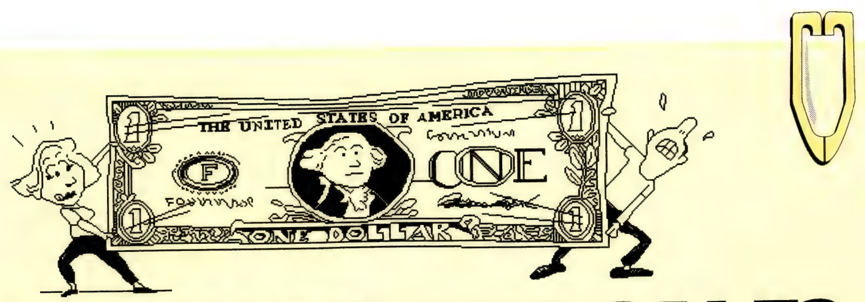
```
pascal void SwitchOutHandler(ThreadID threadBeingSwitched,
void *switchProcParam)
{
  AESwapData* swap = (AESwapData*) switchProcParam;

#ifdef __MWERKS__
  __local_destructor_chain = swap->fStaticChain;
#endif
}


#define ErrCheck(label, result)  if ((result) != noErr) goto label;
```

AEThread

```
pascal long AEThread(AEThreadParam* parms)
{
  AppleEvent    event;       // Original parameters we care about
  AppleEvent    reply;
  AEThreadDesc  * desc;
  OSErr         err;
  OSErr         procErr;
  AESwapData    swap;

  event = *parms->event;      // copy these into our own stack frame
  reply = *parms->reply;
  desc  =  parms->desc;

#if 0
  myTopHandler = gTopHandler;// Save global failure handler
  gTopHandler  = nil;         // don't let failures propagate outside
#endif

  ErrCheck(punt, err = SetThreadSwitcher(kCurrentThreadID,
                     SwitchInHandler,  &swap, true));
  ErrCheck(punt, err = SetThreadSwitcher(kCurrentThreadID,
                     SwitchOutHandler, &swap, false));
  ErrCheck(punt, err = AESuspendTheCurrentEvent(&event));

  parms->result = noErr;      // Let caller know we're ready

  // At this point, we need to let our caller return

  while (desc->holder != kNoThreadID)
  {
    YieldToThread(desc->holder);
  }

  // We are now on our own

  procErr = err = CallAEEventHandlerProc(desc->handler,
                        &event, &reply, desc->refcon);

    // Since the event was suspended, we need to stuff the error code ourselves
    // note that there's not much we can do about reporting errors beyond here

  err = AEPutAttributePtr(&reply, keyErrorNumber,
                        typeShortInteger, &procErr,
                        sizeof(procErr));

#if qDebug
  if (err)
    ProgramBreak("\pAEPutAttributePtr failed installing error
code - very bad");
#endif

  err = AEResumeTheCurrentEvent(&event, &reply,
                      kAENoDispatch, 0);   // This had better work

#if qDebug
  if (err)
    DebugStr("\pAEResumeTheCurrentEvent failed - very bad");
#endif

#if 0
  gTopHandler = myTopHandler;   // Restore global failure handler
  myTopHandler = nil;           // Keep terminator from firing handlers
#endif

punt:
  parms->result = err;
  return nil;
}

#pragma segment Spawn
```

SpawnAEThread

```
pascal OSErr SpawnAEThread(const AppleEvent *event,
                    AppleEvent *reply, long handlerRefcon)
{
  AEThreadParam param;

  param.event  = event;
  param.reply  = reply;
  param.desc   = (AEThreadDesc*) handlerRefcon;
  param.thread = kNoThreadID;

  if (!param.desc)
  {
    param.result = paramErr;
  }
  else
  {
    // make sure no-one else is trying to start a handler
    while (param.desc->holder != kNoThreadID)
    {
      YieldToAnyThread();
    }

    if ((param.result =
         GetCurrentThread(&param.desc->holder)) == noErr)
                                          // Grab the semaphore
    {
      param.result = NewThread(kCooperativeThread,
                 (ThreadEntryProcPtr) &AEThread,
                 &param,
                 param.desc->stackSize,
                 param.desc->options,
                 nil,
                 &param.thread);
```



SOFTWARE FOR SALE?
NO ADVERTISING BUDGET?

Try listing your product in MacTech Magazine's Mail Order Store. Classified advertising at a cost effective rate.

For more information, call:
Voice: 310/575-4343 • Fax: 310/575-0925
AppleLink, GEnie & America Online: MACTECHMAG
CompuServe: 71333,1064 • Internet: marketing@xplain.com

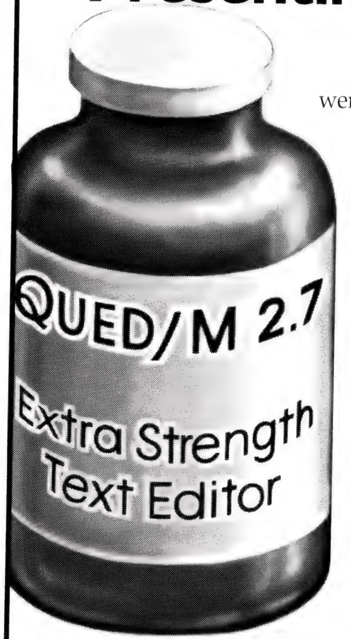
MacTechMAGAZINE™
FOR MACINTOSH PROGRAMMERS & DEVELOPERS

```
    if (param.result == noErr)
    {
      param.result = 1;

      do
      {
        YieldToThread(param.thread);
      }
      while (param.result == 1); // Wait for thread
                                  // to pick up parameters
    }
  }

  param.desc->holder = kNoThreadID; // release any claims we have
}

return param.result;
}
```

FUTURES.C

```
/*
   File:       Futures.c

   Copyright:  © 1993 by Apple Computer, Inc., all rights reserved.
               © 1993-1994 by Steve Sisak, all rights reserved.

*/

#include "Futures.h"
#include <Threads.h>
#include <GestaltEqu.h>

pascal OSErr  IsFutureBlock(AppleEvent* message);
pascal OSErr  DoThreadBlock(AppleEvent* message);
pascal OSErr  DoThreadUnblock(AppleEvent* message);

#define kAERefconAttribute        'refc'
#define kAENonexistantAttribute   'gag!'
#define kImmediateTimeout         0

struct ThreadList
{
  short     numThreads;
  ThreadID  threads[1];
};

typedef struct  ThreadList ThreadList,
                *ThreadListPtr, **ThreadListHdl;

#define sizeofThreadList(numthreads) \
  (sizeof(ThreadList) + ((numthreads)-1)*sizeof(ThreadID))

typedef pascal OSErr (*AESpecialHandlerProcPtr)
                        (AppleEvent *theAppleEvent);

enum {
  uppAESpecialHandlerProcInfo = kPascalStackBased
      | RESULT_SIZE(SIZE_CODE(sizeof(OSErr)))
      | STACK_ROUTINE_PARAMETER(1,
SIZE_CODE(sizeof(AppleEvent*)))
};

#if USESROUTINEDESCRIPTORS
typedef UniversalProcPtr AESpecialHandlerUPP;

#define CallAESpecialHandlerProc(userRoutine,           \
      theAppleEvent, reply, handlerRefcon)              \
```

Presenting the extra-strength text editor.



If you thought previous versions of QUED/M were powerful, wait until you program with QUED/M 2.7, loaded with these new pain-relieving features:

- Integrated support for THINK Project Manager™ 6.0 & 7.0
- THINK™ debugger support
- CodeWarrior™ support (now it's easier than ever to program for the Power Macintosh®!)
- MPW ToolServer™ support
- PopUpFuncs™ support
- Frontier™ Do Script support

For quick relief, call 800-309-0355 today.

Of course, QUED/M 2.7 still has all the features that make it easier to use than any other text editor:

- Macro Language lets you automate tedious tasks
- Search and Replace through multiple unopened files and using GREP metacharacters
- File comparisons using GNU Diff
- Unlimited undos and redos
- 10 Clipboards that can be edited, saved, and printed
- Customizable menu keys
- Text folding
- Display text as ASCII codes
- Plus many more features!

Get even more relief: Try QUED/M 2.7 now for just $69! You'll get a 30-day money back guarantee too! Call now to order. 800-309-0355.

QUED/M is a trademark of Paragon Concepts, Inc. All other products are trademarks or registered trademarks of their respective holders.

107 S. Cedros Ave • Solana Beach, CA 92075 • Tel (619) 481-1477 • Fax (619) 481-6154

NISUS Software Inc.

```
        CallUniversalProc((UniversalProcPtr)(userRoutine), \
        uppAESpecialHandlerProcInfo, (theAppleEvent),      \
        (reply), (handlerRefcon))
#define NewAESpecialHandlerProc(userRoutine)               \
        (AESpecialHandlerUPP)NewRoutineDescriptor(         \
                                  (ProcPtr)(userRoutine),  \
        uppAESpecialHandlerProcInfo, GetCurrentISA())
#else
//typedef AESpecialHandlerProcPtr AESpecialHandlerUPP;
typedef ProcPtr AESpecialHandlerUPP;

#define CallAESpecialHandlerProc(userRoutine,              \
        theAppleEvent, reply, handlerRefcon)               \
      (*(userRoutine))((theAppleEvent), (reply), (handlerRefcon))
#define NewAESpecialHandlerProc(userRoutine)               \
      (AESpecialHandlerUPP)(userRoutine)
#endif

AESpecialHandlerUPP gDoThreadBlockUPP = nil;
AESpecialHandlerUPP gDoThreadUnblockUPP = nil;
AESpecialHandlerUPP gIsFutureBlockUPP = nil;
```

InitFutures

```
pascal OSErr InitFutures(void)
{
  OSErr  err;
  long aResponse;

  gDoThreadBlockUPP = NewAESpecialHandlerProc(DoThreadBlock);
  gDoThreadUnblockUPP= NewAESpecialHandlerProc(DoThreadUnblock);
  gIsFutureBlockUPP = NewAESpecialHandlerProc(DoThreadBlock);

  err = Gestalt(gestaltThreadMgrAttr, &aResponse);

  if (err == noErr)
    err = AEInstallSpecialHandler('blck', gDoThreadBlockUPP,
                                                   false);
  if (err == noErr)
    err = AEInstallSpecialHandler('unbk',
                                   gDoThreadUnblockUPP, false);

  return err;
}
```

DoThreadBlock

```
pascal OSErr DoThreadBlock(AppleEvent* message)
{
  OSErr      err;
  DescType   actualType;
  Size      actualSize;
  ThreadListHdl threadList;
  ThreadID   currentThread;

  // Apparently the current thread needs to access some information, which
  // is really a future. We need to see if there is already a list of threads
  // blocked on this message. If there isn't, create an empty list. Add
  // the current thread to the list. Sleep the current thread.

  err = GetCurrentThread(&currentThread);

  if (err == noErr)
  {
    err = AEGetAttributePtr(message, kAERefconAttribute,
                typeLongInteger, &actualType,
                (Ptr) &threadList, sizeof(threadList),
                &actualSize);

    if (err == errAEDescNotFound
         || err == noErr
        && !threadList)
```

NEW **absoft** NEW

development tools and languages

C++

for Power Macintosh

- Native Power Macintosh C++ compiler
- ANSI and K&R C
- 100% Plum Hall Validated
- Templates
- High Speed Math Library
- Fx™ multi-language debugger
- Power-Link — Absoft's native linker
- Apple's MPW development environment
- Link compatible with Absoft's F77 SDK
- Open, flexible development environment

Available Now: $399.00 Complete

Tel: (810) 853-0050 • Fax : (810) 853-0108 • AppleLink: absoft • Internet: c@absoft.com

```
    {
      // If we can't find a waiting thread list, then create one containing
      // just ourself and put it back in the message

      threadList = (ThreadListHdl) NewHandle(sizeofThreadList(1));

      err = MemError();

      if (err == noErr)
      {
        (**threadList).numThreads = 1;
        (**threadList).threads[0] = currentThread;

        err = AEPutAttributePtr(message, kAERefconAttribute,
                    typeLongInteger,
                    (Ptr) &threadList, sizeof(threadList));
      }
    }
    else if (err == noErr)
    {
      // Otherwise just append ourself onto the existing list

      short numWaiting = (**threadList).numThreads;

      SetHandleSize((Handle) threadList,
                    sizeofThreadList(numWaiting+1));

      err = MemError();

      if (err == noErr)
      {
        (**threadList).threads[numWaiting] = currentThread;
        (**threadList).numThreads = ++numWaiting;
      }
    }
  }

  // If there was an error don't block

  if (err == noErr)
    err = SetThreadState(currentThread, kStoppedThreadState,
                         kNoThreadID);

  return err;
}
```

DoThreadUnblock

```
pascal OSErr DoThreadUnblock(AppleEvent* message)
{
  OSErr            err;
  DescType         actualType;
  Size             actualSize;
  ThreadState      blockedThreadState;
  ThreadListHdl    threadList;

  // This message has just turned real.  If there is a list of threads blocked
  // because they tried to access the data, walk through the list of blocked
  // threads waking and deallocating the list element as you go.

  err = AEGetAttributePtr(message, kAERefconAttribute,
              typeLongInteger, &actualType,
              (Ptr) &threadList, sizeof(threadList),
              &actualSize);

  if (err == errAEDescNotFound)
  {
    // It's possible that this unblocking handler will get called for ALL replies
    // to apple events, not just futures.  If that's the case, then getting
    // the above error is not really an error.  Clear it and just return.

    err = noErr;
  }
  else if (err == noErr && threadList)
  {
    // If there's a waiting list, make all threads in it ready for execution
    // We won't report errors inside the loop because:
    //    1) one of the threads might have been disposed of
    //    2) there's nothing we can do to recover at this point
    //    3) the important thing is to wake up all remaining threads.

    ThreadID*   nextThread = (**threadList).threads;
    short     numWaiting = (**threadList).numThreads;

    while (--numWaiting >= 0)
    {
      ThreadID blockedThread = *nextThread++;
      OSErr   err1 = GetThreadState(blockedThread,
                                   &blockedThreadState);

      if (err1 == noErr
          && blockedThreadState == kStoppedThreadState)
      {
        err1 = SetThreadState(blockedThread,
                        kReadyThreadState, kNoThreadID);
      }
    }

    // Now free the list handle (and take it out of the refCon just to be safe)

    DisposeHandle((Handle) threadList);

    threadList = nil;

    err = AEPutAttributePtr(message, kAERefconAttribute
                typeLongInteger,
                (Ptr) &threadList, sizeof(threadList));

  }

  return err;
}
```

BlockUntilReal

```
pascal OSErr BlockUntilReal(AppleEvent* message)
{
  OSErr      err;
  AERecord   nonExistentParameter;

  err = AEGetParamDesc(message, kAENonexistantAttribute,
                    typeAERecord, &nonExistentParameter);

  if (err == errAEDescNotFound)
    err = noErr;

  return err;
}
```

IsFuture

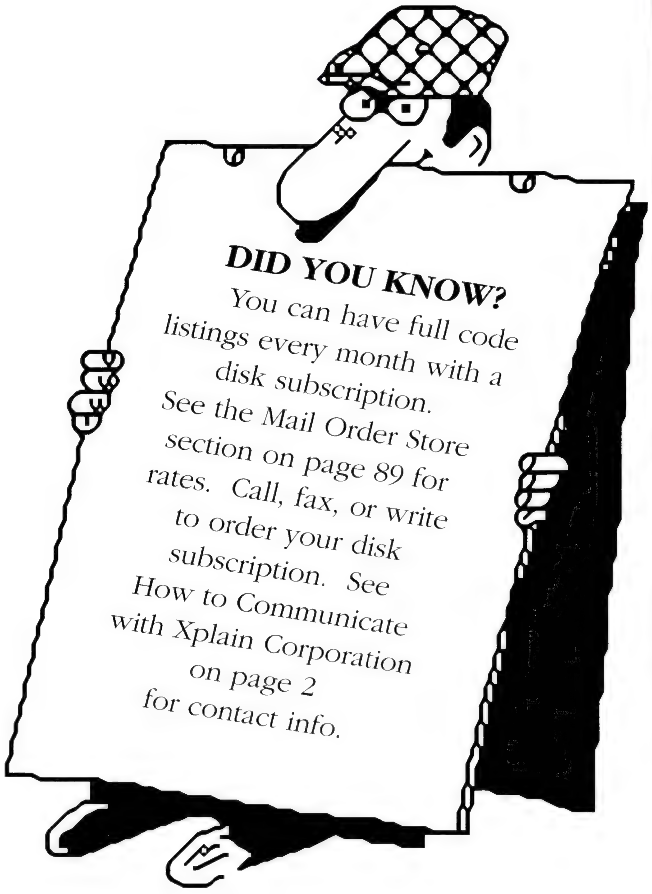
```
pascal Boolean IsFuture(AppleEvent* message)
{
  OSErr                err;
  AESpecialHandlerUPP oldBlockingProc;
  Boolean              isFuture = false;

  err = AEGetSpecialHandler('blck', &oldBlockingProc, false);

  if (err == noErr)
  {
    err = AEInstallSpecialHandler('blck', gIsFutureBlockUPP,
                               false);

    if (err == noErr)
    {
      if (BlockUntilReal(message) == errAEReplyNotArrived)
      {
        isFuture = true;
      }

      err = AEInstallSpecialHandler('blck', oldBlockingProc,
                               false);
    }
  }

  return isFuture;
}
```

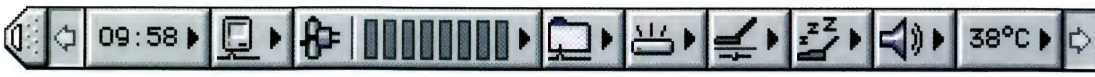
IsFutureBlock

```
pascal OSErr IsFutureBlock(AppleEvent* /*message*/)
{
  return noErr;
}
```

Ask

```
pascal OSErr Ask(AppleEvent* question, AppleEvent* answer)
{
  // Send the question with an immediate timeout.

  OSErr err = AESend(question, answer, kAEWaitReply,
                  kAENormalPriority,
                  kImmediateTimeout, nil, nil);

  if (err == errAETimeout)
    err = noErr;

  return err;
}
```


DID YOU KNOW?
You can have full code listings every month with a disk subscription.
See the Mail Order Store section on page 89 for rates. Call, fax, or write to order your disk subscription. See How to Communicate with Xplain Corporation on page 2 for contact info.



By Mike Blackwell, mkb@cs.cmu.edu

Writing Control Strip Modules

Extend the strip with your own items

When Apple introduced the PowerBook 500 and Duo 280 series computers, they also released the latest version of System 7. Along with the usual new goodies that accompany a system release this included the Control Strip extension. The Control Strip is a long thin window which floats above all other applications on the desktop, always easily available to the user. When retracted, the Control Strip tucks out of the way in a corner of the desktop. When extended, it presents a row of separate chiclet tiles, each displaying some aspect of system status, such as remaining battery life or the state of file sharing. Some of the tiles allow the user to click on them to quickly change the state of the system through a popup menu or dialog box.

Each of the status tiles in the Control Strip is controlled by a chunk of code called a Control Strip Module (specifically, a resource of type 'sdev'), which is generally bundled into a corresponding module file. The user can control which modules are available simply by adding or removing the module files from the Control Strip Modules folder in the system folder. At boot time, the Control Strip extension loads and initializes all of the modules found in this folder.

Control strip modules are fairly easy to write. Since the Control Strip extension itself deals with all of the low level interface to the system, the module programmer is freed from the usual minutiae of Macintosh programming such as toolbox initialization, event loops, grafports, and the likes. However, there are still some tricky areas: resource loading, global memory, loading and saving user preferences, and balloon help, for example. In this article we will use MPW to develop a simple module which displays a rudimentary clock, to learn just what makes a Control Strip module tick.

“Control strip modules are fairly easy to write ... since the Control Strip extension itself deals with all of the low level interface to the system ...”

Before we get started, there are a few caveats to keep in mind:

- Space in the Control Strip is a very limited resource, and should not be needlessly cluttered with modules that might be better implemented as applications. The Control Strip should be reserved for information that the user may need convenient access to at any time.
- Since Control Strip modules are always “running,” they should minimize their impact on system performance by

Mike Blackwell – Mike organizes international symposiums, travels around, and figures out how to program things before Apple gets around to documenting them. He's also been known to do the occasional piece of contract work. Given that chose to go traveling without sending us a bio to fill this space, he's in no position to dispute our claim that he spends most of his time working for a huge unnamed charitable foundation, handing out large grants to starving software entrepreneurs.

consuming as little memory and as few processor cycles as possible.
- Finally, the current release of the Control Strip extension only runs on the PowerBook machines.

If you would like to play with Control Strip on a desktop Mac, you have two options. If you have access to the Control Strip extension (on the System 7.5 installation disk, for example), Rob Mah has produced a nice little patcher program which will modify Control Strip to run on all platforms (by deleting 'sdev' resource number -4064 from the extension, which Control Strip uses as a "PowerBook-only" flag). Also, Sigurdur Asgeirsson has written a shareware Control Strip work-alike called Desktop Strip, which will run on all Macs and use most Control Strip modules (some nice ones are provided). These programs are available at finer archive sites everywhere, try:

```
ftp://mac.archive.umich.edu/
    mac/system.extensions/cdev/controlstrippatcher2.0.sit.hqx
ftp://mac.archive.umich.edu/
    mac/system.extensions/cdev/desktopstrip1.0.sit.hqx
```

They are also available on this issue's code disk and online sites (see p. 2).

Structure of a Module

All code for the module is contained in a code resource of type 'sdev' in the module file. Typically a module file has only one 'sdev' resource corresponding to one module. If a module file has more than one 'sdev' resource, then each is loaded as a separate module, though this gives the user less flexibility in mixing and matching modules. One possible use for multiple modules in one file would be to have different versions of a similar module, only one of which will load depending on the user's hardware configuration. As we will see shortly, during the initialization process a module can decide for itself if it should stay loaded.

Besides the 'sdev' resource, a module file will typically contain a couple of string and picture resources, plus the usual version, icon, and bundle resources to help the user distinguish the file from a generic document. Unless otherwise required, the programmer should number all resources in the range 256 – 32767 to avoid conflicts with other system resources.

In our example module, SimpleCSClock.r is a Rez file which contains all of the resources for the module except for the 'sdev' resource, which will be produced by the C compiler. It starts with a version and signature resource, with information to be displayed by the Finder's Get Info command. The signature ('cs!C' in this case) is also used as the file's creator ID and as a Preferences key, and should be unique. If you are planning to distribute a module widely, its signature should be registered with Apple to prevent conflicts with other applications.

Next is a 'PICT' resource which is a picture of a small right-pointing arrow. This will be drawn in the right-hand side of our tile, to indicate to the user that the module provides a popup menu. A 'MENU' resource defines that menu, followed by a couple of strings and string lists which will be described later. Finally, a 'BNDL', 'FREF', and icon family define the icons for the file that the user will see from the Finder.

The header file SimpleCSClock.h defines the various resource and item numbers in a common location for both the Rez and C programs.

Module Initialization and Global Memory

The Control Strip communicates to the module through a single entry point at the beginning of the module's 'sdev' resource. The prototype for this interface is

```
pascal long ControlStripModule(long message, long params,
        Rect *statusRect, GrafPtr statusPort);
```

`message` signifies which action the Control Strip wishes the module to perform, from which the module can then dispatch the appropriate routines. These actions include module initialization, display management, queries for module features or display size, periodic "tickles," and a couple of others.

`parms` is generally used to store a handle to a parameter or global variables needed by the control strip. The `statusRect` and `statusPort` variables are used when drawing the control strip tile.

In addition to this message interface, the Control Strip extension also provides the module with a handful of utility toolbox routines which simplify many actions common to most modules (the utility routine names all begin with the letters "SB" – until just before its release, Control Strip was called Status Bar).

Control Strip sends `sdevInitModule` as the very first message a module receives. This happens as the control strip extension loads modules from the modules folder at boot time. `sdevInitModule` asks the module to determine if it can run on this particular platform (generally using various calls to Gestalt to query for necessary features), and then initialize its internal state. If the initialization returns a value less than zero to the Control Strip (because it cannot run on this platform or something failed during initialization), then the module is unloaded and not installed in the Control Strip.

In our example, the Initialize() routine is dispatched on receipt of the initialize message (note that `params` will always be 0 for this message). After checking the machine features, a chunk of memory and a handle to it is allocated to hold all of the global variables that the module will need to maintain its state. These variables are defined in the Globals structure at the beginning of the program. If the initialization proceeds successfully, then the handle to the globals is returned to the Control Strip. On all subsequent message calls to the module the Control Strip will pass this handle in `params`, so the globals will always be accessible.

The module file will not be open after the first initialize message has been processed, so any resources that the module needs must be loaded from the file during initialization. This is accomplished with toolbox variants of GetResource. Once each resource is loaded in to memory, its handle is stashed in the globals structure, and the resource is detached so it will always be available. One reason why the module file is not left open after initialization is to conserve power on PowerBooks. If the module were to make GetResource calls on a regular basis it would keep the hard disk turned on a lot.

Once the resources are loaded, the `Initialize()` routine loads and sets any previously saved user's preferences. This is accomplished using two of the Control Strip's utility routines. First, `SBGetDetachedIndString()` is called to get the string item corresponding to the name of our preferences from the recently detached 'STR#' resource `helpStrings`. This name is then passed to `SBLoadPreferences()` to read the module's preferences in to the `SavedSettings` structure. The first field of the preferences structure is checked against 4-byte module signature to make sure these preferences *really* belong to this module. Finally, the user's preferences are copied into the globals structure (in this case, there is only one preference: whether to display the time in 12- or 24-hour format).

Maintaining The Display

Once the module has been initialized and loaded, it can go about its business of maintaining its status display in its tile. This is primarily accomplished through two messages from the Control Strip: `sdevPeriodicTickle` and `sdevDrawStatus`. For these two messages (and all others as well), the Control Strip passes the handle to our globals structure in `params`, a pointer to the drawing rectangle in `statusRect`, and a pointer to the GrafPort in `statusPort` (which typically you won't need to access directly).

Note that each time a message is processed after initialization, the first thing the message handler does is save the state of the globals structure handle, and then lock the structure down so it won't move while the message is being handled. At the completion of the message, the handle's state is restored. We don't simply use HLock()/HUnlock(), because modules can be *reentrant* – it is possible for the Control Strip to call the module's message handler before a previous message has completed. If this happens and we inadvertently unlock the handle underneath another message's feet, all hell could break loose. For this same reason, it's best not to store variables like parameter blocks in the globals structure, since two messages could be trying to modify them at the same time. Allocate these variables on the stack as needed (by declaring them within the scope of the handler routine), or if they must be global, add an in-use flag.

The module performs most of its work in the `sdevPeriodicTickle` message. This message is called periodically, but there is no guarantee about how often or when it will be called. The `sdevPeriodicTickle` message gives the module a chance to check whether its status display is out of date, and if so to update it. Since our clock module updates rather slowly (once every minute), there's no need to check if the time has changed every single time the tickle message is called – once every couple of seconds is adequate and saves wasting cycles needlessly. To accomplish this, we use the `nextTick` global variable to signal when we need to check if the display is out of date. If the system clock (returned by TickCount()) is less than `nextTick`, then we return from the message right away without any further processing.

Otherwise, we call `UpdateTime()`, which checks to see if the time has changed since it was last displayed, and stores the time to be displayed in the globals structure. If the time has changed, the tile rectangle is erased and `DrawDisplay()` is called to update the display. `DrawDisplay()`, in turn, converts the current time in to a string, selects the proper font, sets the drawing point within the statusRect, and calls DrawString() to draw the time of day in the module's tile. `DrawDisplay()` then draws the small right-pointing arrow picture just to the right of the time string, which indicates that a popup menu is available. Finally, `nextTick` is set for two seconds from the current time.

After drawing all of the information in the `statusRect`, the tickle message handler compares the width of what it has just drawn (returned by `DrawDisplay()`) to the width of what it had drawn the previous time through (and saved in the width field of the globals structure). If the width has changed then the `width` field is updated, and `sdevResizeDisplay` bit is set to be returned to the message call. This bit signals to the Control Strip to shrink or grow the module's tile appropriately so all of the data is visible in the tile.

When the Control Strip is informed that the module's tile width has changed, it will next send the module an `sdevGetDisplayWidth` message request. Since the module is maintaining that width anyway, there's no need to recompute it, and the module simply returns the width field from the globals structure.

The other message which maintains the module's displays is `sdevDrawStatus`. This message is sent by the Control Strip when the display needs to be redrawn, such as for a window update or display hilite, and requests the module to draw its display in its tile. In our case, we do this by calling the `DrawTime()` routine. Since the module will always be displaying the same information it has already displayed, the width will not have changed and the return from `DrawDisplay()` can be ignored.

User Interaction

Not all modules require input from the user – they may simply display status information – but our clock module allows the user to select the format of the time presentation. The user can select either a 12- or 24-hour display format by clicking on the module's tile and making a menu selection from the popup menu that appears. The currently selected option is flagged with a bullet (Figure 1).

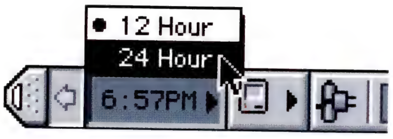


Figure 1

The module informed the Control Strip that it wanted to process mouse clicks in its tile by setting the sdevWantMouseClicks and sdevDontAutoTrack bits in its reply to an sdevFeatures message. Whenever the Control Strip detects a click in the module's tile, it does two things: first, it highlights the tile by darkening its background and shifting the display slightly for a 3-D effect, and then it sends the module an sdevMouseClick message. If the sdevDontAutoTrack feature bit had not been set, then the module would not receive the sdevMouseClick message until after the mouse was clicked and then released in the tile, causing the tile to act as a button instead of a menu title.

Our module responds to the sdevMouseClick message by calling its HandleMouseClick() routine. This routine first checks the current display mode, and marks the appropriate selection in the menu with a bullet character (conveniently pre-defined as sdevMenuItemMark). It then displays the popup menu by calling the Control Strip utility routine SBTrackPopupMenu(). Once a menu item has been selected, the appropriate display mode is flagged in the global structure. If the display mode was changed, then the reply to the sdevMouseClick message has the appropriate bits set to inform the Control Strip that the new user preference needs to be saved and that the balloon help message has changed. DrawDisplay() is called with a flag to inhibit actual drawing to compute the width of the new display. Similar to the tickle message, if the width has changed then the global width is updated and the sdevResizeDisplay reply bit is set to inform Control Strip.

User Preferences

The Control Strip provides the module with a very nice mechanism to save its state between shutdown and restart. This state is stored as a 'pref' resource in the Control Strip Preferences file in the system Preferences folder.

At some convenient point after a module has requested to save its settings (by setting the sdevNeedToSave bit in the reply to an sdevMouseClick message, for example) it will receive an sdevSaveSettings message. Control Strip typically waits until some other process has powered up the hard disk, or during shutdown time, to send this message. When our module receives this message, it dispatches its SavePreferences() routine. This allocates memory for the SavedSettings structure, which for our module has only two fields: an identifying signature, and the boolean display mode we want to save. The signature is just a safety feature to make sure the module doesn't get confused later by inadvertently trying to load some other module's incorrectly stored resource, and we'll set it to our file creator type. The display mode is just copied from the current mode in the global structure.

Once the settings structure is set up and filled in, the module needs to pick a name for the resource. For convenience, we already predefined this name as an item in the help strings 'STR#' resource, so the module calls SBGetDetachedIndString() to retrieve the name. Then the module calls SBSavePreferences() to save the settings structure as a 'pref' resource in the preferences file.

On startup these steps are basically performed in reverse to load and set the previously saved settings, as described in the Module Initialization section.

Balloon Help

The Control Strip allows each module to provide simple balloon help to the user. If balloon help is turned on and the module has set the sdevHasCustomHelp feature bit, then the Control Strip will send the module an sdevShowBalloonHelp message when the mouse is sitting in the module's tile. The module can then call SBShowHelpString() with a descriptive string to be displayed to the user (Figure 2). There is no easy way to provide more detailed help for things like menu items.

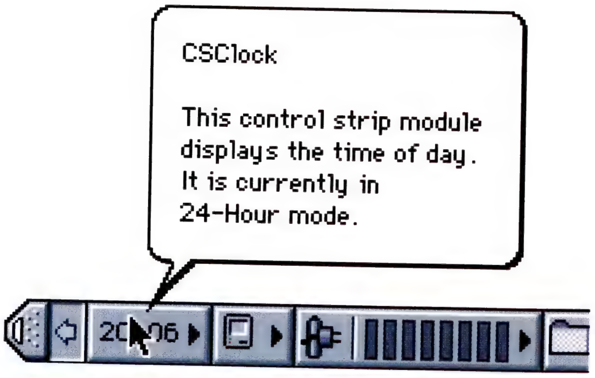

CSClock
This control strip module displays the time of day. It is currently in 24-Hour mode.
2006

Figure 2

If the module changes its mode while handling a tickle or mouse click message, it can set the sdevHelpStateChanged bit in its reply, telling the Control Strip to issue a new sdevShowBalloonHelp message to cause the help string to change.

To give the user more information about the module file, our module has a Finder help string resource (a 'STR ' with an ID of -16397). If the user double clicks on the module's icon from the Finder, the Finder will display a dialog box containing this string, telling the user what the module is and how to use it. We've also set a Help Manager resource pointing to this same string. The idea is that balloon help from the Finder will display the string, but this does not currently work because the Finder doesn't know about 'sdev' files so it always displays a generic message. But it only costs a few bytes, and maybe one day the Finder will do the right thing.

Additional Resources

Hopefully, this article has given you enough information to go out and write your own Control Strip modules. The full documentation for the Control Strip API can be found, oddly enough, in Chapter 5 of Apple's PowerBook 520/520c/540/540c Developer Notes. You can find a copy on any recent Develop Bookmark CD (a very worthwhile addition to your programming library to complement your MacTech magazines). It will also soon be released as a tech note.

The Control Strip header file ControlStrip.h can be found on Apple's ETO CD #15 and later. If you don't have access to the "official" version, Rob Mah provides a home grown version along with his patcher. Note that there is a slight inconsistency between Apple's and Mah's definitions of the return bits for tickle and feature messages. Apple defines the bit number (for example, sdevHasCustomHelp = 2) while Mah defines the bit position (sdevHasCustomHelp = (1 << 2)). The code in this article uses Apple's conventions.

Many thanks to Steve Christensen at Apple, Control Strip's author, for reviewing this article.

SimpleCSClock.h

```
#define kSignature          'cs!C'

#define kArrowPictID        256

#define kConfigMenuID       256
#define     k12HourCmd        1
#define     k24HourCmd        2

#define kHelpStringsID      256
#define     kPrefNameStr      1
#define     kHelp12StringStr  2
#define     kHelp24StringStr  3
```

SimpleCSClock.c

```
/* SimpleCSClock - A simple control strip clock module

   By Mike Blackwell, mkb@cs.cmu.edu

*/

#include <GestaltEqu.h>
#include <Fonts.h>
#include <Memory.h>
#include <Packages.h>
#include <Resources.h>
#include <Strings.h>
#include <SysEqu.h>
#include <ToolUtils.h>
#include "ControlStrip.h"
#include "SimpleCSClock.h"


// How often to check if the display needs to be updated (in ticks)
#define INTERVAL                (2 * 60)     // 2 seconds

// Display font information
#define DISPLAY_FONT            monaco
#define DISPLAY_FONT_SIZE       9
#define DISPLAY_FONT_FACE       0
#define DISPLAY_MARGIN          3     // Blank space to left and right of text

typedef struct Globals {
  PicHandle   arrowPicture;    // Picture to show we have a popup menu
  short       arrowWidth,
              arrowHeight;     // Size of arrow
  short       fontHeight;      // Ascender height of display font
  Handle      helpStrings;     // Balloon help strings for each state
  MenuHandle  configMenu;      // Menu to select display options
  int         width;           // Width of display
  Boolean     show12Hour;      // True for 12 hour display
  int         displayTime;     // Time that is currently being displayed
                               //  (in minutes since midnight)
  unsigned long nextTick;      // When to next update the display
} Globals, *GlobalPtr, **GlobalHandle;


typedef struct SavedSettings {
  OSType      signature;       // Signature to verify that prefs are for
                               //  this module
  Boolean     show12Hour;      // True for 12 hour display
} SavedSettings;
```

Prototypes

```
long    Initialize(void);
void    CleanUp(GlobalHandle globHand);
long    HandleMouseClick(GlobalPtr globPtr, Rect *statusRect);
short     SavePreferences(GlobalPtr globPtr);
Boolean   UpdateTime(GlobalPtr globPtr);
int       DrawDisplay(GlobalPtr globPtr, Rect *statusRect,
                      Boolean drawit);
Boolean   CheckFeatures(void);


pascal long main( long message,
                  long params,
                  Rect *statusRect,
                  GrafPtr statusPort)
{
#pragma unused(statusPort)
  char        savedState;
  GlobalPtr   globPtr;
  long        result;
  int         current_width;
  Str255      helpString;

  if (params > 0) {                          // If we have globals allocated,
    savedState = HGetState((Handle)params);
                                             //  save the handle state,
    HLock((Handle)params);                   //  lock the handle to the globals,
    globPtr = *(GlobalHandle)params;         //  and point to globals directly
  }

  result = 0;                        // Return zero for unknown messages

  switch (message) {

    case sdevInitModule:                     // Initialize the module
      result = Initialize();
      break;

    case sdevCloseModule:                    // Clean up before being closed
      CleanUp((GlobalHandle)params);
      params = 0L;                           // Handle is gone now
      break;

    case sdevFeatures:                       // Return feature bits
      result = (1 << sdevWantMouseClicks) | \
            (1 << sdevDontAutoTrack)     | \
            (1 << sdevHasCustomHelp);
      break;

    case sdevGetDisplayWidth:                // Return display width
      result = globPtr->width;
      break;

    case sdevPeriodicTickle:                 // Periodic tickle when nothing
```

```
                                        // else is happening
      // Time to update display yet?
      if (TickCount() >= globPtr->nextTick) {
        if (UpdateTime(globPtr)) {      // Check if time has changed
          EraseRect(statusRect);        // Yep, erase the old
          current_width = DrawDisplay( globPtr,
                                       statusRect,
                                       true);// And draw the new
          // If the display width changed, let the control strip know
          if (globPtr->width != current_width) {
            globPtr->width = current_width;
            result = (1 << sdevResizeDisplay);
          }
        }
        globPtr->nextTick = TickCount() + INTERVAL;
      }
      break;

    case sdevDrawStatus:                // Update the display
      (void)DrawDisplay(globPtr, statusRect, true);
      break;

    case sdevMouseClick:                // User clicked on the module's
                                        // display area in the status bar
      result = HandleMouseClick(globPtr, statusRect);
      break;

    case sdevSaveSettings:              // Save changed settings
      result = SavePreferences(globPtr);
      break;

    case sdevShowBalloonHelp:           // Display custom balloon help
      SBGetDetachedIndString(&helpString,
        globPtr->helpStrings,
        globPtr->show12Hour ? kHelp12StringStr : kHelp24StringStr);
      SBShowHelpString(statusRect, &helpString);
      break;
  }

  if ((long)params > 0)                 // If we have globals allocated,
    HSetState((Handle)params, savedState);
                                        // restore the locked/unlocked state

  return(result);
}
```

Initialize

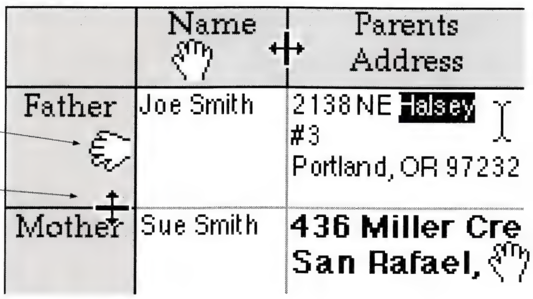
```
long Initialize(void)
{
  long        result;
  GlobalPtr      globPtr;
  GlobalHandle globHand;
  FontInfo    fontInfo;
  Str255      prefsResourceName;
  SavedSettings **preferences;

  result = -1;                          // Assume failure

  if (!CheckFeatures())
    return(result);


  if (! (globHand =
          (GlobalHandle)NewHandleClear(sizeof(Globals))))
    goto done;                          // Allocate the globals


  HLock((Handle)globHand);              // Lock the globals while using them
  globPtr = *globHand;                  // and get a pointer to them

// Load and detach the 'up arrow' picture

  if (! (globPtr->arrowPicture = GetPicture(kArrowPictID)))
    goto done;


  DetachResource((Handle)globPtr->arrowPicture);

// Compute size of arrow picture

  globPtr->arrowHeight =
     (**globPtr->arrowPicture).picFrame.bottom
     - (**globPtr->arrowPicture).picFrame.top;
  globPtr->arrowWidth =
     (**globPtr->arrowPicture).picFrame.right
     - (**globPtr->arrowPicture).picFrame.left;

// Compute size of display font

  TextFont(DISPLAY_FONT);
  TextSize(DISPLAY_FONT_SIZE);
  TextFace(DISPLAY_FONT_FACE);
  GetFontInfo(&fontInfo);
  globPtr->fontHeight = fontInfo.ascent;

// Load and detach the configuration menu

  if (! (globPtr->configMenu = GetMenu(kConfigMenuID)))
    goto done;


  DetachResource((Handle)globPtr->configMenu);

// Load and detach the help strings

  if (! (globPtr->helpStrings =
                      Get1Resource('STR#', kHelpStringsID)))
  goto done;


  DetachResource(globPtr->helpStrings);

// Get the module's saved preferences, if any, and configure the module

  SBGetDetachedIndString(&prefsResourceName,
                          globPtr->helpStrings, kPrefNameStr);
  if (! SBLoadPreferences(&prefsResourceName,
                          (Handle *)&preferences)
      &&
        ((**preferences).signature == kSignature)) {
      globPtr->show12Hour = (**preferences).show12Hour;
  }

  globPtr->nextTick = 0;               // Do first update right away
  globPtr->width = 0;
  globPtr->displayTime = 0;            // Haven't displayed a time yet

  HUnlock((Handle)globHand);           // Unlock the globals

  result = (long)globHand;             // Return the handle to the
                                       // globals as the result

done:
  return(result);                      // Return either a handle or
                                       // an error code
}
```

CleanUp

```
void CleanUp(GlobalHandle globHand)
{
  GlobalPtr      globPtr;

  if ((long)globHand <= 0) return;

  HLock((Handle)globHand);
  globPtr = *globHand;

  if (globPtr->arrowPicture)
      DisposeHandle((Handle)globPtr->arrowPicture);

  if (globPtr->configMenu)
      DisposeMenu(globPtr->configMenu);

  if (globPtr->helpStrings)
      DisposeHandle(globPtr->helpStrings);

  DisposeHandle((Handle)globHand);
}
```

StoneTable ™ a drop-in replacement library for the List Manager

titles
variable height rows
variable width columns
default titles of letters or numbers
move / copy row / column
sort rows / columns
resize column / row
“LDEF-like” custom
drawing support
in use since early ’92


Name
Parents Address
Father
Joe Smith
2138 NE Halsey #3 Portland, OR 97232
Mother
Sue Smith
436 Miller Cre San Rafael,

edit cell / title in place
different font per cell
appl. defined cell validation
hide rows / columns
fore / back color per cell
32K text per cell
> 32K non-text per cell
scroll cell text

StoneTablet Publishing
P.O. Box 12665
Portland, OR 97212
(503) 287-3424
CIS: 70303,2546
70303.2546@compuserve.com

All the functions of the List Manager
plus support for those listed above and more.
Simple conversion from the List Manager.

Libraries available for
Think C, Think Pascal, MPW C $150 each
International shipping (US Airmail) $10
Payment only by check in US$
No royalty fees for applications
TCL/MacApp not required

HandleMouseClick

```
long HandleMouseClick(GlobalPtr globPtr, Rect *statusRect)
{
  short       menuItem;
  long        result;
  int         new_width;
  Boolean     mode_changed;

  // Check off the appropriate items in the popup menu

  if (globPtr->show12Hour) {
    SetItemMark(globPtr->configMenu, k12HourCmd,
                sdevMenuItemMark);
    SetItemMark(globPtr->configMenu, k24HourCmd, noMark);
  } else {
    SetItemMark(globPtr->configMenu, k24HourCmd,
                sdevMenuItemMark);
    SetItemMark(globPtr->configMenu, k12HourCmd, noMark);
  }

  result = 0;

  // Display the popup menu

  menuItem = SBTrackPopupMenu(statusRect,
                              globPtr->configMenu);

  // Handle the menu selection

  mode_changed = false;

  switch (menuItem) {
    case k12HourCmd:
      if (!globPtr->show12Hour) {
        globPtr->show12Hour = true;
        mode_changed = true;
      }
      break;

    case k24HourCmd:
      if (globPtr->show12Hour) {
        globPtr->show12Hour = false;
        mode_changed = true;
      }
      break;
  }

  // If the mode changed, calculate new display width
  // and let CS know what happened

  if (mode_changed) {
    result = (1 << sdevNeedToSave)
           | (1 << sdevHelpStateChange);
    new_width = DrawDisplay(globPtr, statusRect, false);
    if (globPtr->width != new_width) {
      globPtr->width = new_width;
      result |= (1 << sdevResizeDisplay);
    }
  }

  return(result);
}
```

SavePreferences

```
short SavePreferences(GlobalPtr globPtr)
{
  short           result;
  SavedSettings **preferences;
  Str255          prefsResourceName;

  preferences = (SavedSettings**)NewHandle(
                              sizeof(SavedSettings));

  if (! (result = MemError())) {  // Allocate a block to hold the settings

    // Include a signature to verify it's ours
    (**preferences).signature = kSignature;

    (**preferences).show12Hour = globPtr->show12Hour;

    // Get the name of the preferences resource
    SBGetDetachedIndString(prefsResourceName,
                           globPtr->helpStrings, kPrefNameStr);

    // Save the settings in the Control Strip's preferences file
    result = SBSavePreferences(prefsResourceName,
                               (Handle)preferences);

    DisposeHandle((Handle)preferences);        // Get rid of the block
  }

  return(result);
}
```

UpdateTime

```
// Compute the current time in seconds since midnight. If it's changed since we
// last displayed the time, return true.

Boolean UpdateTime(GlobalPtr globPtr)
{
  unsigned long now;
  int           dispTime;

  GetDateTime(&now);                    // Get seconds since epoch

// Compute minutes since midnight
  dispTime = (now % (60 * 60 * 24)) / 60;

// Has the time changed yet?
  if (dispTime != globPtr->displayTime) {
    globPtr->displayTime = dispTime;    // Yes
    return(true);                       // Need to update display
  } else {
    return(false);
  }
}
```

NumStr

```
// Stuff the ascii string representing the number in to the destination string.
// Number range is 0 - 99. If pad is true, pad with a zero if necessary.
// Return pointer to end of string.

char *NumStr(char *dest, int num, Boolean pad)
{
  if (num < 0)
    num = 0;
  if (num > 99)
    num = 99;
  if (num >= 10) {
    *dest++ = (num / 10) + '0';
    num %= 10;
  } else if (pad) {
    *dest++ = '0';
  }
  *dest++ = num + '0';
  *dest = '\0';
  return(dest);
}
```

DrawDisplay

```
// Draw the time specified in displayTime. If drawit is false, don't actually
// do any drawing. Returns the width of the display

int DrawDisplay( GlobalPtr globPtr, Rect *statusRect,
                 Boolean drawit)
{
  int        result;
  char       buf[10],
             *bptr;
  int        hours, minutes;
  Boolean    afternoon;
  int        width, offset;
  Rect       arrowRect;

  result = 0;

  hours = globPtr->displayTime / 60;
  if (globPtr->show12Hour) {
    afternoon = (hours >= 12);
    if (afternoon)
        hours -= 12;
    if (hours == 0)
        hours = 12;
  }
  minutes = globPtr->displayTime % 60;

  bptr = buf;

  if (globPtr->show12Hour) {
    bptr = NumStr(bptr, hours, false);
    *bptr++ = ':';
    bptr = NumStr(bptr, minutes, true);
    *bptr++ = (afternoon) ? 'P' : 'A';
    *bptr++ = 'M';
    *bptr = '\0';
  } else {
    bptr = NumStr(bptr, hours, true);
    *bptr++ = ':';
    bptr = NumStr(bptr, minutes, true);
    *bptr = '\0';
  }
  c2pstr(buf);

// Draw the time string a little away from the right edge and centered vertically
// Compute top/bottom margin. -1 is tweak to make it look just right...
  TextFont(DISPLAY_FONT);
  TextSize(DISPLAY_FONT_SIZE);
  TextFace(DISPLAY_FONT_FACE);
  if (drawit) {
    offset = (statusRect->bottom - statusRect->top
             - globPtr->fontHeight) / 2 - 1;
    MoveTo(statusRect->left + DISPLAY_MARGIN,
          statusRect->top + offset + globPtr->fontHeight);
    DrawString(buf);
  }

  width = DISPLAY_MARGIN + StringWidth(buf) - 1;

// Draw the right arrow to show that the module has a popup menu
  if (drawit) {
    arrowRect.left = statusRect->left + width;
    arrowRect.right = arrowRect.left + globPtr->arrowWidth;
    arrowRect.top = statusRect->top +
            (statusRect->bottom - statusRect->top
             - globPtr->arrowHeight) / 2;
    arrowRect.bottom = arrowRect.top + globPtr->arrowHeight;
    DrawPicture(globPtr->arrowPicture, &arrowRect);
  }

  width += globPtr->arrowWidth;

  return(width);
}
```

CheckFeatures

```
// Check if this machine is capable of running this control strip (probably
// by using Gestalt). This simple example will run on any Mac, so just return
// true.

Boolean CheckFeatures(void)
{
  return(true);
}
```

Is it time to renew your MacTech Magazine subscription? Check your mailing label to see your expiration date. If you need to renew, write, call or fax the MacTech Magazine main offices.

TOOLS OF THE TRADE

By Andy Dent, A.D. Software, Ballajura, Western Australia

Object Master 2.5 Universal

A slightly dazzled review...

I tend to develop a love-hate relationship with ACI products, and Object Master is no exception. Maybe it's a French culture clash with my British origin. Object Master has a few petty flaws that annoyed me to the "up all night to teach this sucker a lesson" level. Redeeming itself, Object Master is also full of features that elicited ooh's and aah's when I demonstrated it to a friend. I spent 10 minutes with the OM demo on the CodeWarrior CD and was hooked. I hope I can show you why.

Perspective

Immediate apologies to MPW mavens and MacApp-ites. I'm neither and so I can't fully evaluate those features of Object Master. What I'd like to do is show anyone just how much time Object Master can save you in learning class libraries and developing within complex frameworks. I haven't been this excited about an OOP tool since I first saw the SmallTalk/V object browser. That may be because I'm trying to learn two different frameworks (zApp and PowerPlant) at the same time, neither of which comes with an object browser.

Note: in the rest of the article I'll follow the Object Master convention of using the words *method* (C++ member function) and *field* (C++ data member). This terminology seems to be also used by most OOF (OO Folks) when discussing language-independent issues. In the interests of brevity, I'll also use the abbreviation OM for Object Master.

Reasons to rush out and buy Object Master

OM is a tool to let you browse and manage both OOP and non-OOP code written in C, C++, Pascal and Modula-2. OM drives your compilers and linkers directly, removing the need to use the compiler IDE. It will save you a massive amount of time managing code in a complex framework, and especially learning a new framework:

- The SmallTalk-like Browser lets you edit classes and methods in a very point-and-click environment, focusing on just what you're changing.
- The tree browser lets you display both methods and fields for any or all classes.
- The tree browser can be truncated to show you just a part of the tree, and you can turn off the Multiple Inheritance lines (useful for the PowerPlant forest).
- You can have a number of different tree and class browsers visible at once, showing you portions of the classes in varying detail.
- Subclass definitions can be generated with all the methods automatically generated for you to fill in, and override.
- C++ users can view classes from the viewpoint of different access methods. (eg: what fields do you see as a Friend class?)
- The editor provides syntax highlighting, including choice of style and color.

Andy Dent – Andy comes to us from the ranks of the former Software Frameworks Association (SFA), and has enjoyed using Object Master to help himself master new big chunks of object oriented code, like PowerPlant. You can reach Andy on the Internet: dent@iinet.com.au, or on CompuServe: 100033,3241, A.D. Software

- An MS-Word style of line selection with a "backwards arrow" at the start of the line lets you avoid tendonitis-inducing triple-clicks.
- Think Reference is fully integrated to lookup definitions.
- MPW 411 files can be generated so you can keep your documentation in parallel with your source. (For the ignorant, whose ranks I recently departed, 411 files are indexed text files used for documentation, like an MPW Think Reference.)
- Function calls can be automatically completed, and the case corrected, from your own declarations, Think Reference or MPW 411 documents.
- SourceServer and Projector are fully supported, plus arbitrary read-locking of files.
- If you produce .SYM files (MPW and CodeWarrior) you can browse disassembled code in parallel with your source.
- Remote compilation is supported via ARA, using MPW ToolServer. (I know one user who swears by this, for working from home.)
- Resources compiled into a project can be browsed and ResEdit launched with direct selection of the resource.
- You can script in either MPW or AppleScript.
- Macro facilities provide a "fill-in-the-blanks" method for tabbing through complex declarations and common idioms. If you are heavily into patterns and idioms, this could be a great time-saver (or help those laboring under corporate style-guides).

Getting Started and the Project Environment

Where's the "I'm Stuck!" Documentation?

I've seen a couple of people very strongly criticize OM in the MacDev forum on CompuServe. After running into problems with my initial PowerPlant parsing, I can understand why. The Reference Manual is generally very good, but it lacks a companion Getting Started piece. I spent a few painful hours deducing the following simple facts:

- CodeWarrior integration is not complete (who's at fault?)
- If you get syntax errors at the end of a file, it's almost certainly some weird compiler-specific construct earlier in the file and you can probably fix it by defining a Null or Identity Macro.
- The AppleScript menu at the end of the menu bar isn't documented because it's added by the scripts in the Startup Folder.
- If a Think Pascal tree is flat and you can't see fields or class definitions, you have an interface file which is saved as a Think document, not as text.

Project Management

The OM environment is project-based. You can create a project directly from existing compiler projects, from defaults supplied with OM, from your own stationery, or from scratch. A project includes settings such as the macro definitions used to parse C++ files, and style settings for the various editing windows.

I'd like to put in a cheer for one of my favorite prejudices here – ACI has made good use of Balloon Help to assist in learning the complex menu structure and the occasional iconic buttons. Balloons are also used to describe the short-cut clicks that can be used in various scrolling list panes.

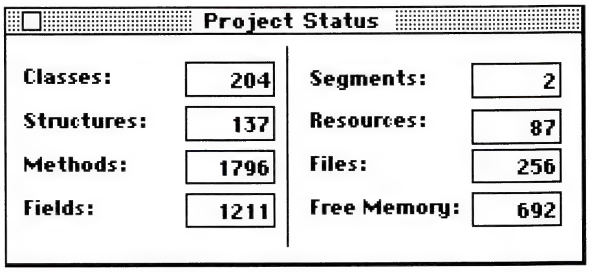
As you work in the different Project windows, you will often make use of secondary menu bars within the windows. This is an optional violation of the Apple Human Interface Guidelines. There is a Preferences setting to place these menus at the end of the main menu, but you would need a two-page monitor to see them! I found the secondary menus usable, although a little slower than a quick mouse to the top of the screen. In contrast to MS Windows packages I've used, the secondary menus are more comprehensible and hence faster than some vast array of iconic buttons.

Think Pascal Project Support

Creating a project direct from a Think Pascal project is very straightforward, but it may not work! The manual neglects to inform you that OM doesn't understand the Think document format. This is compounded by OM's failure to warn you that it has not been able to find the expected text file. OM deduces the class names from the implementation files, but you need to save a copy of TCL.p as text before you can see a proper class tree hierarchy, or any class definitions. Assuming your Pascal project has been built, the resources will be visible in OM, through a Resource Map window.

Think C Project Support

The addition of external support in Symantec C++ for Mac v6 allowed OM to directly drive the Think compilers in building projects. Loading from a project seems to work but neglects to load the resource file – you have to add that manually with the Add Files dialog. I loaded the complete TCL 1.1.3 tree and a sample Marksman-generated TCL application with no complaints. If you are only using the Think C+Objects compiler, you can set this preference in OM to speed the parsing process.


Project Status

Classes:	204	Segments:	2
Structures:	137	Resources:	87
Methods:	1796	Files:	256
Fields:	1211	Free Memory:	692

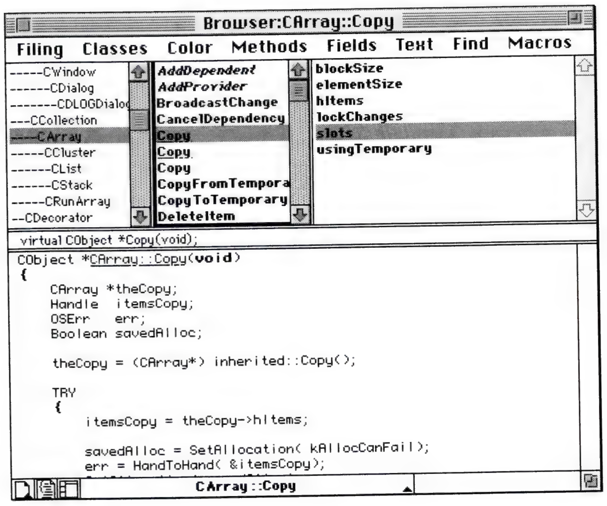
Figure 1. Project Status window after loading the Marksman TCL project.

The Class Browser

If you've used a Smalltalk browser such as SmallTalk/V, or the 4D Insider tool, then the Browser window is familiar. The three scrolling lists across the top (or down the left-hand side) list the Classes, Methods of the current Class, and Fields of the Current class. The editing area contains either a single Method or Class definition.

OM allows you to have a number of Browser windows open at once, showing different aspects of the class hierarchy and with different settings (such as showing inherited Methods or Fields). The Browser is the first window in which you'll encounter OM's secondary menu bars.

C++ template classes are shown with a trailing † symbol on the class and method names. Multiply-inherited classes are shown with a trailing ° symbol on the class name. Global functions and structures are gathered into pseudo-classes and prefixed with a • symbol. (This is how you can use OM to browse a completely non-oop program.)



```
-----CWindow
------CDialog
-------CDLOGDialog
---CCollection
----CArray
-----CCluster
------CList
------CStack
-----CRunArray
--CDecorator
```

```
AddDependent
AddProvider
BroadcastChange
CancelDependency
Copy
Copy
Copy
CopyFromTempora
CopyToTemporary
DeleteItem
```

```
blockSize
elementSize
hItems
lockChanges
slots
usingTemporary
```

```
virtual CObject *Copy(void);
```

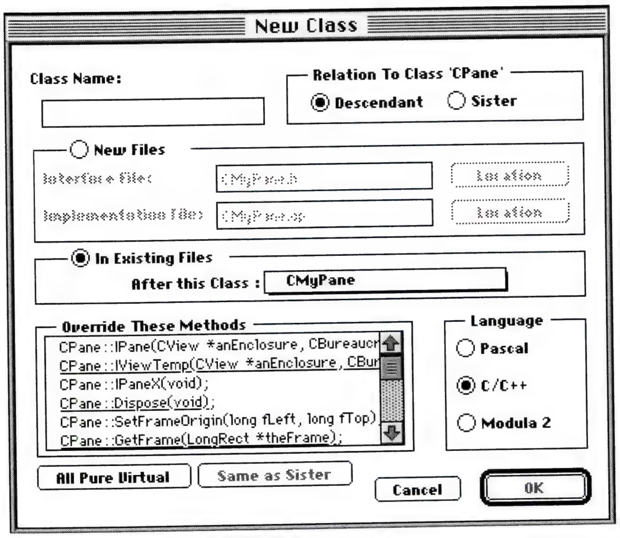
```
CObject *CArray::Copy(void)
{
    CArray *theCopy;
    Handle  itemsCopy;
    OSErr   err;
    Boolean savedAlloc;

    theCopy = (CArray*) inherited::Copy();

    TRY
    {
        itemsCopy = theCopy->hItems;

        savedAlloc = SetAllocation( kAllocCanFail);
        err = HandToHand( &itemsCopy);
```

Figure 2. Object Browser with Show Inherited Methods and the parent implementation CObject::Copy selected.

Editing bit-by-bit

One major benefit of using a Class Browser is focusing on the class definition or method implementation that you are writing. The OM Browser lets you very quickly navigate to the method or a parent class's implementation (using the Show Inherited Methods option). Figure 2 shows how I've been able to go directly to the Copy method implemented by a parent class, looking at CArray's methods. The "middle" pane that stretches across the center of the browser is a context feedback. When you click on a method name it shows the full signature. When you click on a class you see the inheritance path, including multiple parents (the pane is resizable). Notice the highlighting around the methods list – the panes of the browser are all keyboard-navigable, using command-Tab to switch panes.

If you are jumping between several methods, and lack the screen space for multiple browser windows, you can use a popup menu, on the bottom of the browser window, to recall any of the last ten methods edited.



Class Name:

Relation To Class 'CPane': [✓ Descendant] Sister

New Files
Interface File: CMyPane.h
Implementation File: CMyPane.cp

[✓ In Existing Files]
After this Class : CMyPane

Override These Methods
```
CPane::IPane(CView *anEnclosure, CBureaucr
CPane::IViewTemp(CView *anEnclosure, CBur
CPane::IPaneX(void);
CPane::Dispose(void);
CPane::SetFrameOrigin(long fLeft, long fTop)
CPane::GetFrame(LongRect *theFrame);
```

Language: Pascal [✓ C/C++] Modula 2

Figure 3. New Class Dialog.

Creating Classes with point-and-click

I've always hated the messy business of defining classes that are almost-but-not-entirely-like another class, maybe overriding a couple of parent methods and adding a field or two. The New Class command in the Browser Filing menu simplifies this task enormously. You can either define a straight subclass of the current class, or a "sister" class. For a sister class, the Same as Sister button quickly marks all the methods you need to override to be just like the sister class. You can then mark a few more Methods, or un-mark some to leave them to the parent implementation. The choice of filing is flexible and makes it easy to accumulate a number of small classes in the one file.

"I haven't been this excited about an OOP tool since I first saw the SmallTalk/V object browser."

Following Relationships

When you are trying to learn a class framework, as opposed to adding to it, you can spend an inordinate amount of time yo-yoing up and down the class hierarchy, wondering who implements a method, who calls it and how many classes further down the hierarchy override the method.

The Methods menu has an option to display the inherited methods of the current class. You can also create list windoids of classes that:

- implement the current method (same name, possibly unrelated);
- override the current method;

- call the current method (a text search that includes comments).

Understanding C++ Access Control

In moving from Object Pascal to Think C+Objects to C++, I've often been confused over when and how to use Friend classes (and, to a lesser extent, the other access control levels). The OM Browser allows you change the methods and fields displayed to simulate what would be available from a different Access Level. Normally, when editing, you can see everything (as an Implementor) but you can change this to restrict the list to those available to a Friend or a Client class (only the public methods and fields.)

As a short-cut, when you are editing a method (X), you can click on a different class name and choose the Class Access option See Class from Open Method. OM uses its knowledge of the relationship between the classes to display only the fields and methods visible to method X. This can be a great help at those times when you are getting compiler errors related to class access, but can't quite understand why.

Segments by menu

The Methods menu allows you to display the Segment of Method. You can also change this from the Browser by choosing Filing – Set Method Segment. This invokes a dialog with a popup list of the current segments, and the option to type in a new segment name. OK'ing the dialog will insert #pragma directives where appropriate. For more comprehensive changes over segments, a Segment Map window allows you to drag many methods at once, to a new segment.

The Tree Browser

I was always a big fan of the Think environment's class tree browser, but used to take up most of a monitor displaying it and flipping between classes at opposite ends of the tree. The multiple tree browsers allowed in OM and the pruning capabilities are a great time saver, by comparison.

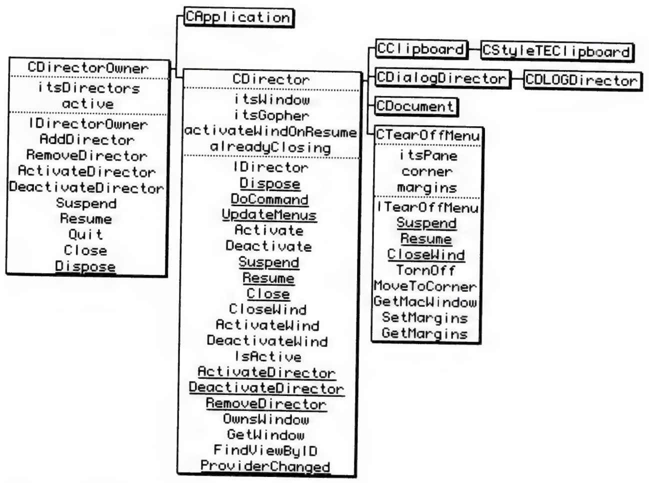
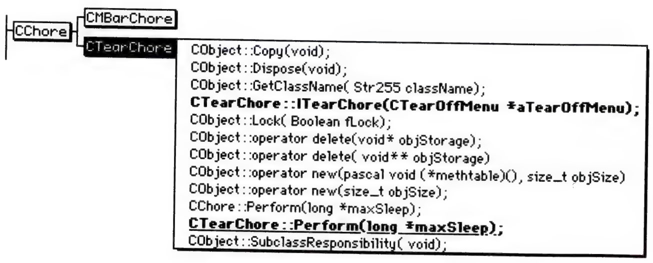


Figure 4. Partial Class Tree, with some Methods and Fields.

Displaying Fields and Methods

Class Trees are a great way to display the inheritance path. When documenting or tracing a class hierarchy, sometimes you want to see what's implemented by a given class. Figure 4 shows a partial Class Tree with the methods and fields displayed on three of the classes. The underlined methods are the virtual methods.

We've already seen the Browser allows you to display inherited fields and methods. The Class Tree nodes can popup either just the methods of the selected class, or you can turn on Add Inherited Members. Figure 5 shows how this gives you a (sometimes massive) menu with class names included and bolding used to highlight classes of the current node.



```
CObject::Copy(void);
CObject::Dispose(void);
CObject::GetClassName( Str255 className);
CTearChore::ITearChore(CTearOffMenu *aTearOffMenu);
CObject::Lock( Boolean fLock);
CObject::operator delete(void* objStorage);
CObject::operator delete( void** objStorage)
CObject::operator new(pascal void (*methtable)(), size_t objSize)
CObject::operator new(size_t objSize);
CChore::Perform(long *maxSleep);
CTearChore::Perform(long *maxSleep);
CObject::SubclassResponsibility( void);
```

Figure 5. Popup menu of Methods, showing Inherited Methods

Focusing on Parts of the Tree

Figure 4 was produced by a couple of steps, starting with clicking on CDialogDirector and choosing Focus on Class. This reduced the class tree to just the parents of that class, and its sister classes at the same level on the tree. To complete the diagram, I then expanded CClipboard and CDialogDirector in one go, with the Expand All Classes command. Unlike the other figures, Figure 4 is not a screen shot but is produced directly as a PICT file with the Tree menu's Save as Pict command. This could be a boon to documentors.

A faster way of producing Figure 4 would have been to click on CDirectorOwner and choose Zoom in Descendants. This reduces the tree to the sub-tree starting at the selected class. Finally, as described below, the User Class attribute can be assigned to any class and the tree reduced to just User Classes and their parents. In common use, you would use User Class to mark your classes as distinct from the MacApp or TCL hierarchy. One major disappointment for me was the Color menu. Assigning colors to classes allows you to sort by color in a Browser window. It would be very nice if the Class window allowed you to prune the tree to just the Blue classes. You could then mark several frameworks in different colors.

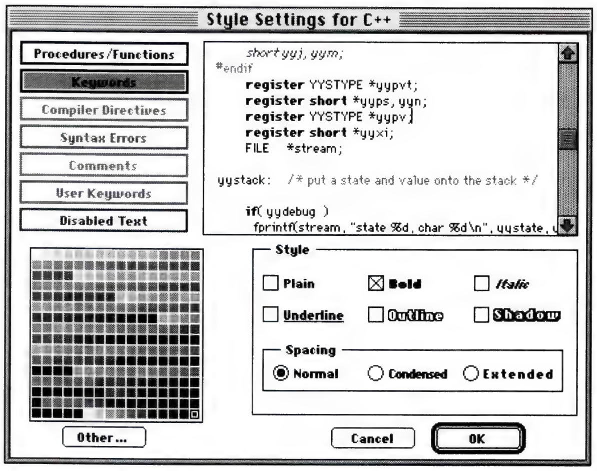
Multiple Inheritance

Multiple Inheritance is supported very cleanly in the class tree windows with lines drawn from all the parent classes. This can become very messy, in a dense forest such as PowerPlant, so the Single Inheritance command is very useful. It suppresses the lines from the additional parent classes. One minor disappointment was that the Focus on Class command only shows the direct parent classes. This would be a lot more useful if it also pulled in the additional parent classes. One way around this, if you want to show a partial tree with the MI parents, is to use the User Class command to mark the classes. You just have to mark the MI parents and the leaf class you are focusing on, then use the Minimal Tree command to reduce the tree to the User Classes and their parents.

Editing

Most of the editing features are available in both the editing area of the Browser windows, and the full File windows used to display an entire source file. The File windows also provide an MPW-style Markers menu, which shows user-selected markers as well as auto-generated markers for each method and structure definition.

Comprehensive search and replace operations are of course available. One nice touch is an effective keyword search, by searching for known entities in the dictionary. This includes class and method names and non-OOP structure names, depending on parsing preferences. I was a little disappointed to see that the grep search only allows MPW syntax and not the standard UNIX characters. However, the graphical constructor of grep queries is a great help.



```
        short yyj, yym;
#endif
        register YYSTYPE *yypvt;
        register short *yyps, yyn;
        register YYSTYPE *yypv;
        register short *yyxi;
        FILE    *stream;

yystack:    /* put a state and value onto the stack */

        if( yydebug )
         fprintf(stream, "state %d, char %d\n", yystate, 
```

Style: [ ] Plain [✓ Bold] [ ] Italic [ ] Underline [ ] Outline [ ] Shadow

Spacing: [✓ Normal] Condensed Extended

Figure 6. C++ Style Settings for Editing Areas.

Keyword Color and Styling

Color and font styling in editors is a contentious issue. Figure 6 shows the range of styling controls. I'm very much a fan of subtle color coding. I like suppressing my comments slightly using a violet shade, so extensive commenting doesn't crowd out the code. Silly errors with nested comments are immediately obvious with such color coding, and basic syntactic errors leap out at you in glaring red. If you tend to be a slightly sloppy typist (or occasionally work with a child on your lap) then error highlighting is a great time saver, compared to running the compiler.

Lookups and Completion

I'm forever looking up templates for complex toolbox functions – sometimes I think I bought my second monitor just to run Think Reference. As well as the common style of command-doubleclick to go directly to a reference, OM has Find and Paste Method Call. This feature looks up definitions from your methods, open 411 files and Think Reference (in that order). Best of all, it is not case-sensitive. For example:

- type getfinfo and hit command-shift-J
- Think Reference is invoked to complete the line to read:

GetFInfo(fileName,vRefNum,&fndrInfo)

C Preprocessor Ignorance and Compiler Specifics

The most confusing issue I encountered was that OM doesn't understand the C preprocessor. This statement seems silly when you start – the syntax highlighting clearly picks out compiler directives and italicizes the False branch in #if statements.

OM does understand the structure of #if statements and has a few predefined True and False constants. It does not understand #define, so if you have any macros that are used in the same sense as normal C++ keywords or functions, then you must define them manually. (Note: ACI say v3 will have full parsing.) The C\C++ submenu of the Settings menu allows you to define four kinds of macros:

- Null Macros (#define Macro);
- Identity Macros (#define Macro(x) x);
- Ignore Macros (#define Macro(x));
- Keyword Macros (redefine C++ and C keywords, from a popup menu).

Something that is very poorly documented is how to cope with compiler extensions such as CodeWarrior's ONEWORDINLINE. Although these are compiler keywords, and not macros, any such extension to C++ v3 syntax must be treated as a macro. As described below, you need to add a number of such macros to successfully parse the PowerPlant classes (and I'm still adding to the list for zApp for Windows).

Apparent bugs in the parsing

```
#if !defined(MC_PLATFORM)
# define PLATFORM_MACINTOSH
# define MC_PLATFORM  PLATFORM_MACINTOSH
#endif
```

Defaults to a true branch, so you would expect PLATFORM_MACINTOSH to be known. However, remember that #define is not really understood by the OM parser, so may think you should define PLATFORM_MACINTOSH as a True & Defined Symbol yourself.

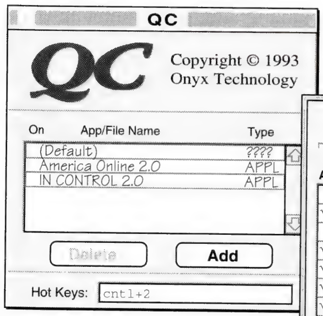
No: this is still not good enough as the OM #define parsing doesn't understand logical operators. Having defined _INTELC32_ as a False Symbol by hand, you would expect the #defines in the following code to be included. They're not, but it doesn't really matter.

QC

NOW SHIPPING

QC: the **Macintosh Testing solution.** Subject your code to **brutal** stress conditions to **make it break** consistantly.

Or use QC during the development cycle to casually detect block boundary overwrites, invalid BlockMoves, writes to location zero, and more… ***saving countless hours of needless debugging.***

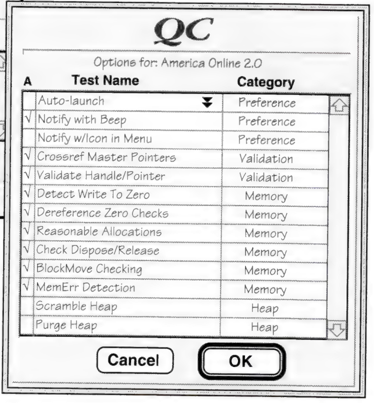


QC

Copyright © 1993 Onyx Technology

On	App/File Name	Type
	(Default)	????
	America Online 2.0	APPL
	IN CONTROL 2.0	APPL

Hot Keys: cntl+2



QC

Options for: America Online 2.0

A	Test Name	Category
	Auto-launch	Preference
√	Notify with Beep	Preference
	Notify w/Icon in Menu	Preference
√	Crossref Master Pointers	Validation
√	Validate Handle/Pointer	Validation
√	Detect Write To Zero	Memory
√	Dereference Zero Checks	Memory
√	Reasonable Allocations	Memory
√	Check Dispose/Release	Memory
√	BlockMove Checking	Memory
√	MemErr Detection	Memory
	Scramble Heap	Heap
	Purge Heap	Heap

Features:

- Works with any program without modification - source not required
- Easy to use: just hit the hotkey.
- Fast heap scramble and purge
- Invalidates all free memory
- Detects runtime block overwrites
- Warns of DisposHandle on resource and ReleaseResource on handles.
- Validates BlockMove destinations
- Sophisticated heap verification
- Powerful API for precision control

$99 Special Introductory Price

30 Day No Questions Asked Money Back Guarantee

Onyx Technology

Onyx Technology
7811 27th Ave W.
Bradenton, FL 34209

Tel: 813.795.7801 **Fax:** 813.792.5152 **AOL:** OnyxTech **ALink:** D2238 **CIS:** 70550,1377

You don't have to worry too much about whether #if or #else branches are taken if the only things bracketed are #defines, as they are ignored. Remember, any preprocessor symbols that have syntactic significance must be manually defined.

```
#if defined(_INTELC32_) || defined(PLATFORM_MACINTOSH)
# define far
# define near
# define huge
# define cdecl
#endif
```

So, after all the above worrying over how to get the preprocessor #if branching correct, we still have to go through and manually add a definition of w_cdecl, to get the OM parser to accept the following line:

```
typedef void        w_cdecl (*MFUNC) (char *msgtext);
```

CodeWarrior Support

OM Universal v2.5 is advertised as supporting CodeWarrior and the on-disk Addendum includes specific instructions. However, I found the project commands to synchronize and load files from projects were disabled. ACI US Technical support (via CompuServe) have confirmed this is due to CodeWarrior still not providing full AppleEvents for external editors.

Parsing Problems

CodeWarrior defines a number of compiler extensions. Most of these are keywords that can be simply added to the OM macro definitions, as shown below. However, I was unable to resolve a way to get Object Master to accept constructs such as the following, where the :__D0 is an indicator that the parameter is passed in a register. One problem is that the macro definitions dialog doesn't allow the colon character as part of a macro name.

```
static long RestoreA4(long:__D0):__D0 = 0xC18C;
```

Defined Ignore Macros for the following (v2.5.2 knows the xxWORDINLINE's):

```
ONEWORDINLINE
TWOWORDINLINE
THREEWORDINLINE
FOURWORDINLINE
FIVEWORDINLINE
SIXWORDINLINE
CallComponentRoutineProc
ComponentCallNow
NewComponentRoutineProc
```

Define Null Macros (in addition to those such as pascal already in a new project):

```
_C_LIB_DECL
_END_C_LIB_DECL
_EXTERN_C
_END_EXTERN_C
THREAD_INLINE_PROC
```

MPW May Rescue Us

When the 68k CodeWarrior MPW tools arrive, missing in CW4, things may change. Given the high level of MPW integration, OM users will probably ignore the CodeWarrior IDE and build directly using the MPW tools. This will be a quick fix to the problems of integrating with CodeWarrior projects, but I hope both options will be available in future release of OM.

Scripting and Customizing

AppleScript Built-In

AppleScript is very tightly integrated into OM, although unfortunately not recordability. There is a Startup Folder which can contain scripts to be executed to customize the OM environment, or perform any other startup action. The initial scripts in this folder add an AppleScript menu to the OM main menu, with a number of useful scripts. eg: List Methods With Parameter… which searches all methods and creates a list windoid of all matches, for the parameter type you enter (warning: choosing something general like Handle takes a few minutes on a IIci).

There are some eight pages of class definitions for AppleScript classes. Between them, they seem to offer control over the content and attributes of most components. The only omission that bothers me is the 411 Documentation files. It would be nice to have scripted retrieval of documentation, to aid in building reference manuals. I assume the MPW experts will be able to cope with this but I feel a little left out.

Executing AppleScripts is very well thought out. Apart from the AppleScript menu, scripts can be entered into the MPW-like Worksheet window and activated with a command-Enter. If you are really in a hurry, any piece of text in an editing window can be executed with the Execute in AppleScript item on the Text menu.

MPW Support

One thing that disappointed me about the tight links between OM and MPW is that you have to specify MPW as your compiler. For example, the editing windows allow invocation of the MPW tools CDent and PasMat. If you have CodeWarrior or Think Pascal selected as your compiler, you have to enter Preferences and change this to MPW Shell or ToolServer before the Reformat command or icon work.

As part of the AppleScript support, OM defines a rich suite of AppleEvents. The long list of bundled MPW Scripts and Tools includes the OM_SendAE tool. This lets you exert the same scripting control from MPW as AppleScript or Frontier.

MPW is OM's native environment. As well as driving the various compiler and linker tools, OM has support for Jasik's Incremental Build System (an incremental linker). Projector support for version control is cleanly provided with a popup menu on the bottom left corner of File and Browser windows.

Stationery and Custom Resources

OM fully supports System 7 stationery settings. In addition to creating Stationery documents, you can put a Project Options file in the Preferences folder to set your standard project options. This includes basic settings as well as automatically including whatever files are part of your Project Options.

For those wishing permanent change to Object Master's defaults and appearances, the elegant solution is the OM Customization file. This is a resource file, initially blank, which is opened straight after the application. A detailed chapter of the manual lists the resource numbers and types you can add to this file, to override the standard ones without fear of conflict.

WINDOWS PLANS

From the number of postings I've seen in comp.lang.c++ asking about object browsers, the following CompuServe posting should come as very good news (I wonder how they ported from a MacApp v2 product?):

23 Aug 1994 3:43:11 From: ACI US Tech Support 76207,1331

We will be releasing a Windows version next quarter. We have just finish the Alpha Phase of testing. The Windows version will support all major compilation systems on the PC. Borland, Microsoft, Symantec, etc.

There will be major feature that will allow the user to do true cross platform development.

– Joe Levenson, ACI US

Finally, a commercial-grade spellchecker toolkit as powerful as the machines your products run on.

(No royalties, accurate suggestions, case and duplicate word checking, American, British and specialty dictionaries, multiple character sets, free updates.)

LexTek International

2255 N. University Parkway, Suite 15, Provo UT 84604 • (801) 375-8332

Programmer Training

MACINTOSH SEMINARS & CONSULTING

Richey Software Training provides professional, *customized* programming seminars and industry consulting.

Rich content & hands-on lab exercises shorten your learning curve. On-site training is convenient – your team eliminates expensive travel costs and consuming down-time.

MAC SEMINARS
- C & C++
- OOP
- MacApp
- PowerPC
- AppleScript
- MPW
- System 7
- Debugging
- SourceBug

"Professional training & consultancy that really hits the mark."

Call today to find out how Richey Software Training delivers professional seminars and consulting nationwide.

RICHEY SOFTWARE TRAINING

Training Mac Programmers Since 1986

707•869•2836

AppleLink: RICHEY.SOFT
INTERNET: 70413.2710@compuserve.com
P.O. BOX 1809, GUERNEVILLE, CA 95446-1809

© 1994 Richey Software Training. All trademarks or registered trademarks are the property of their respective owners. Specifications subject to change.

Pascal → C++

Stuck with Pascal source code?

Want to move to C++?

OP2CPlus is a Macintosh tool for converting Object Pascal source code to C++ source code. It has already been used to translate hundreds of thousands of lines of Object Pascal to C++.

Ease your transition to OpenDoc or PowerPC

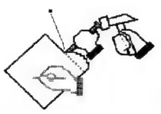
- ❑ Convert in days instead of months
- ❑ Generates C++ classes
- ❑ Works with MacApp 2 or 3
- ❑ Translates Think or MPW Pascal
- ❑ Full ANSI C source code included
- ❑ Just $895

Graphic Magic Inc, 180 Seventh Ave, Suite 201
Santa Cruz CA 95062 Tel (408) 464 1949
Fax (408) 464 0731 AppleLink GRAPHICMAGIC

MacHack'95

June 22-24, 1995 – Michigan
The 10th Annual Conference

Call for Papers



Topics include, but aren't limited to:
- Emerging Technologies...
  OpenDoc/OLE, GX, PowerTalk
- Cross Platform Development/Strategies
- Emerging Platforms...
  Telescript, MagicCap, Newton, etc.
- User Interface Design
- Game Design/Development
- Programming Culture Issues...
  Agents, people?, passing the torch, etc.

Paper presenters get free conference registration and special recognition. Bound proceedings will be distributed to all attendees.

This year's Keynote is Chris Crawford!

Registration Fee: $375 before 1/1/95, Speakers - $275
Attendance is limited, so register early!

For more information or to register, contact:
Expotech at 313-882-1824
1264 Bedford Rd.
Grosse Pointe Park, MI 48230-1116
AppleLink or AOL: Expotech
Internet: expotech@aol.com

Macintosh and Newton are registered trademarks of Apple Computer. All others trademarks are property of their respective holders. MacHack is a trademark of Expotech, Inc. Not affiliated with the MacHax™ Group. All rights reserved.

CONCLUSIONS AND WISH-LIST

Writing this review was a great excuse to go romp through Object Master's exhaustive (and exhausting) feature list. I hope I've convinced a majority of you that the product is worth looking at, and I salute Metrowerks for including a demo version. Since writing the review, I've also used OM to learn the zApp framework. It has been worth buying just for the time saved in this project!

ACI seem to be very interested in maintaining OM and there has been enough public criticism that I would expect the problems with parsing bad code to be fixed very soon (Note: v2.5.2 includes some improvements and is much more stable). ACI's Technical Support in the ACIUS forum on CompuServe is free and fairly quick to respond. Their response to this article helped me clarify a few points about the fixes in v2.5.2 and they've annotated the wish list shown below.

The increased emphasis on C++ should lead them to improve CodeWarrior and general C++ support, and rectify the lack of Getting Started docs. Here's my wish list for features in OM version 2.6 (CMaster users may feel a sense of deja-vu):

High on the v3 list
- Splitter bar or bars in the File editor
- Block commenting with enough intelligence to allow commenting out a block and then reversing the process without losing existing comments

Already in native version, and in next 68k minor release
- Script or other support for adding and removing line feeds to ease transition between Mac and DOS
- Make it possible/easy to copy macro definitions between projects. (Drag-n-drop)

Under Consideration
- Tree browser Focus on Class includes multiple-inherited parents
- Applescript control over 411 Documentation supported as a property of Method and Class AppleScript classes.

Good Ideas
- Enable users of other compilers to run MPW scripts (this would let us use the MPW tools such as Mac2DOS)
- Drag-and-drop in the Segment Window shouldn't require holding down Option to reassign methods to segments
- Smarter indentation control, rather than having to run the external MPW indent
- Tree browser allows restrict by color, not just User Class

ACI US offers a free demo disk of Object Master. You can contact them at ACI US at (800) 384-0010 or (408) 252-4444 *voice*, (408) 252-0831 *fax*, AppleLink D4444, or mail them at ACI US, 20883 Stevens Creek Blvd., Cupertino CA 95014.

By Paul Robichaux, Fairgate Technologies, Harvest, AL

QC: Test for Success

Integrated debugging & stress testing software

Taking the Sting Out Of The "D" Word

Every programmer does it. Experienced programmers usually do less of it than neophytes, and programmers on some platforms do it more than others. What is it? Caffeine consumption? Swearing at operating system designers? Well, besides those... I'm talking about debugging. By some estimates, debugging can take up to 40% of development time, and studies have indicated that there can be as much as a twenty-to-one difference in productivity between a run-of-the-mill debugger and a really skilled debugger.[1]

The Macintosh operating system architecture offers unique challenges to debuggers. The combination of handles and memory-moving traps provide fertile ground for bugs related to the almost Brownian motion of relocatable blocks. Memory-related errors are made more difficult to isolate and fix because writing to memory that belongs to other applications, or writing beyond the bounds of a particular block, doesn't always cause a crash. If a crash does occur, it can often be quite some time after the original bad write, so the cause of the bug can be masked by other paths taken through the source code.

There are a wide range of freely available tools available for memory & resource debugging. They were generally written because their authors needed them, and then saw benefit to giving them away so others could benefit from their work. That was good for everyone. However, the heroes who wrote and distributed them (people like Greg Marriott, who wrote EvenBetterBusError, DoubleTrouble, and DisposeHandle; and Bo3b Johnson, who wrote Leaks, Blat, and ZapHandles) have real jobs, and decided not to make a career of keeping the tools up to date and adding features. For example, Blat doesn't work on very old or very new Mac CPUs. DisposeResource and DoubleTrouble slow system-wide performance since they are always on when installed. Still others reveal faults not only in your code, but in others' as well. That's fine if you have time to deal with other folks' buggy software, but it sometimes gets in the way of debugging your own.[2]

"Since there are several free tools which incorporate various parts of QC's functionality, why would you pay $99?"

Onyx Software has gone the extra mile and built QC™, a control panel which offers a variety of tools designed to help you build rock-solid code. QC allows you to selectively turn on a wide range of stress-testing features (including almost all of those provided by the above-mentioned tools) on a per-application

Paul Robichaux – Paul Robichaux leads a dual life: by day, he builds Unix and Windows NT applications for a major computer company. During the rest of the time, he builds custom-engineered Mac applications and WWW pages for clients in a wide variety of fields. He welcomes reader e-mail at paul@fairgate.com, or visit his WWW page at http://www.iquest.com/~fairgate.

basis. QC also includes a versatile API which allows you to embed calls to QC's engine within your own code. The QC documentation says that QC is "specifically designed to make memory-related errors reproducible," and it excels at that task.

What QC Is For

QC is designed to perform two key functions. First, it helps you find memory and resource-handling errors during development. Second, it provides a simple way for quality assurance testers to put unusual stress on an application to flush out any remaining errors or performance problems.

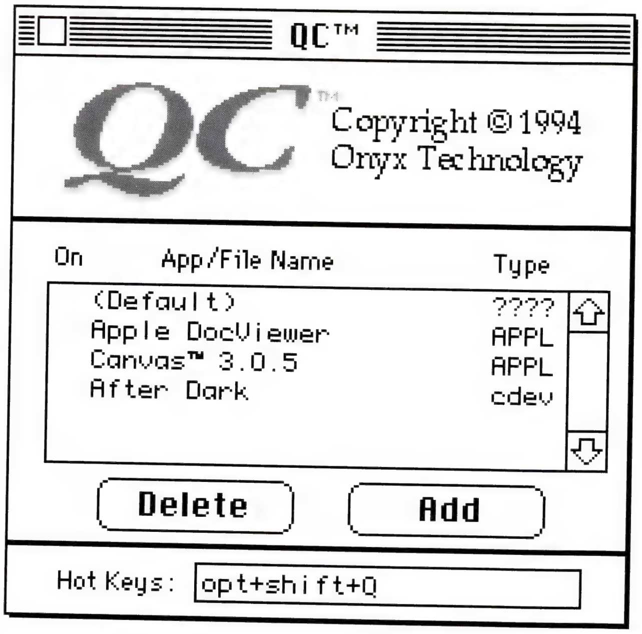
Most developers are already familiar with the two primary types of debuggers: low-level and source-level. Both types allow you to interactively examine the contents of memory and processor registers. The primary difference between them is how you interact with the code you're debugging. Low-level debuggers, like Macsbug, typically require that you work with the disassembled contents of memory. You set breakpoints by absolute or relative memory address, and the contents of structures referenced by pointers may not always be identifiable by type. Source-level debuggers show you the source statements of your programs and allow you to set breakpoints and examine variables in the context of that, and some even allow you to execute function calls without interrupting execution.

QC doesn't really fall into either category. It's not a debugger in and of itself; it's a debugging aid. You can't manipulate or examine the contents of memory using QC. Instead, it can force your code to drop into the installed low-level debugger when it encounters an error, or it can simply beep. With the provided API, you can perform any operation expressible in code when an error occurs. The API also allows you to turn on a particular kind of debugging for any section in your code, which makes the beep option more useful than you might otherwise suspect.

Getting Started With QC

As you'd expect, installing the QC control panel is very, very simple: drag it to your system folder and reboot. When you open it the first time you'll have to personalize it; after that, you'll see the control panel's main window:

The center list, called the "target list," allows you to set debugging checks on an individual basis. Each application can have a unique combination of the 15 distinct checks that QC knows how to perform, and you can modify which checks should be performed while the application is being QCed. If you prefer, QC can automatically start testing when the application or code resource under study starts executing. The block at the window's bottom allows you to specify a hot key to toggle QC at any time. Pressing the hot key causes QC to start checking the currently active application; pressing it again turns checking off. You can also use the auto-launch setting to start QC's tests when an application starts or a code resource is loaded.

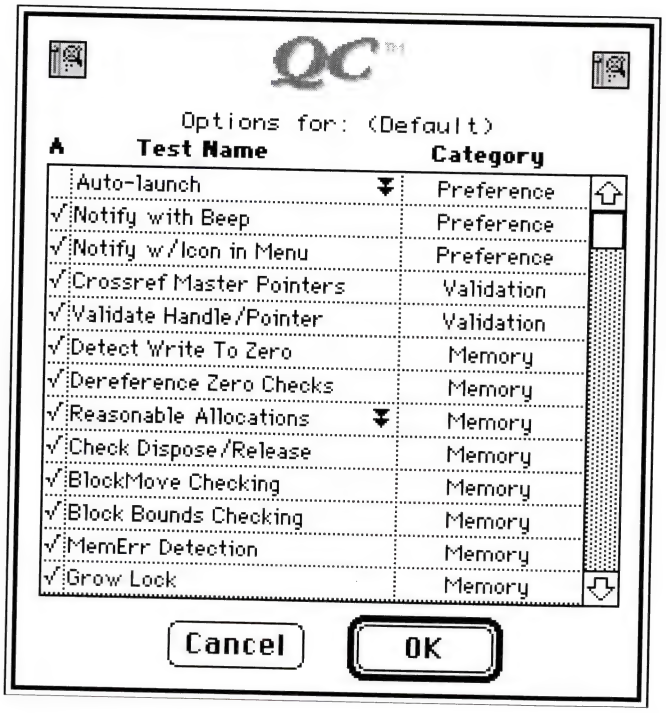


On	App/File Name	Type
	(Default)	????
	Apple DocViewer	APPL
	Canvas™ 3.0.5	APPL
	After Dark	cdev

Hot Keys: opt+shift+Q

Figure 1: The QC control panel main window.

Double-clicking one of the items in the target list brings up a dialog of test options. List items with downward-pointing triangles have associated parameters which you can set either via the control panel or the API routines.



Options for: (Default)

A	Test Name	Category
	Auto-launch ▼	Preference
√	Notify with Beep	Preference
√	Notify w/Icon in Menu	Preference
√	Crossref Master Pointers	Validation
√	Validate Handle/Pointer	Validation
√	Detect Write To Zero	Memory
√	Dereference Zero Checks	Memory
√	Reasonable Allocations ▼	Memory
√	Check Dispose/Release	Memory
√	BlockMove Checking	Memory
√	Block Bounds Checking	Memory
√	MemErr Detection	Memory
√	Grow Lock	Memory

Figure 2: QC's test configuration dialog.

You can choose QC's behavior when it detects an error – QC will either beep or dump you into the installed low-level debugger. If you're using the API routines, QC will not generate

Choosing an installer program should be *easy*

Easy for your users	DragInstall's unique drag-and-drop interface makes installing your software a piece of cake for your users.
Easy for you	DragInstall's Builder utility allows you to create complex installers in minutes not days. And without the need to learn a complicated scripting language.
Easy on your wallet	DragInstall's one-time fee allows you to distribute an unlimited number of installers. No yearly renewals, no royalties, no hassle!
Easy as DragInstall	See for yourself just how easy DragInstall is. Contact us for a free demo disk and information kit

New in DragInstall 1.5.3

- New compression algorithm saves 15-20% more space than previous versions.
- International templates allow building of French, German, and Italian installers.
- New external allows creation of aliases for installed files.

Ray Sauers Associates, Inc.

1187 Main Avenue, Suite 1B • Clifton, NJ 07011-2252 • Voice: 201-478-1970 • Fax: 201-478-1513 • AppleLink: D1922 • AOL: SAUERS

an error on its own; instead, it'll return an error code so that you can do whatever reporting or cross-checking is appropriate. If you choose, your callback routine can let QC handle the error itself.

What QC Knows How To Check

QC has a total of 15 different tests. Any combination of tests can be enabled for any application or code resource. You can run the tests on both application and system memory and heaps. All but two of the tests run on 68k and PowerPC Macs; the "invalidate free memory" and "block bounds checking" tests are automatically disabled when running under the Modern Memory Manager.

I found that the tests tended to fall into one of three categories: memory-handling tests, Toolbox use tests, and stress tests. Some combinations of tests are particularly effective at flushing out errors.

Memory and Resource Tests

These tests verify that your code is staying inside its own heap and address space. In particular, these tests are useful for catching code which uses handles or pointers after they've been disposed.

Cross-reference master pointers

This test checks that every relocatable handle in your heap is pointed to by a master pointer within a master pointer block. As you might imagine, it's not good if you have a relocatable block in your heap which isn't pointed to by a master pointer.

Validate handles/pointers

This test validates each handle or pointer passed to Memory Manager calls to ensure that their addresses fall within the application's heap and that the block to which the pointer or handle points is of the correct type. This is perhaps the most useful single test in QC's arsenal, since it can detect a multiplicity of sins.

Detect writes to location zero

This very useful test sets a trap for code which uses dangling pointers or handles. When enabled, QC places a special tag value at memory location zero; at each trap call to see, QC checks to ensure that location zero still contains that tag.

Dereferencing zero

Like the "detect writes to location zero" option, this option salts location zero with a special value. In this case, the special value is one that causes a bus error (on 68020 or later CPUs) or an odd address error (on 68000 CPUs.) When your code attempts to dereference that tag, as it would when mistakenly trying to use a purged handle, you'll immediately be dumped into the installed low-level debugger.

Toolbox Use Tests

The Mac OS offers developers a very powerful set of tools. This power carries with it some danger, because many Toolbox routines fail when given bad parameters, and some calls are dangerous when misused. These tests check for valid parameters and proper use of calls. They are not a substitute for using correct prototypes in your code.

Reasonable allocations

This test checks parameters to NewHandle() and NewPtr() to make sure that they're positive values less than or equal to a user-specified size. On some compilers, it also works with library routines like malloc() which call the Toolbox routines to allocate memory.

Check dispose/release calls

Everyone was a Mac newcomer at some time, and this test catches a classic newcomer's mistake: calling ReleaseResource() on a handle or DisposeHandle() on a resource. It takes some practice to make sure to do the right thing with handles which point to resources, and this test provides welcome reinforcement.

BlockMove bounds checking

The Toolbox BlockMove() call can fail rather spectacularly if you call it with addresses which span multiple blocks in the heap. This test makes QC check that the start and end addresses of the source and target blocks don't cross block boundaries.

Block bounds checking

Some languages offer compile- or run-time bounds checking, but C doesn't. Programs which write outside the bounds of an allocated block or array will sometimes crash, but sometimes they won't. To make it easier to find the problem, this test causes QC to add a tag value to each allocated block. Your application doesn't see the tag, but QC can. When it detects that the tag has been overwritten, it generates an error. This test is a lifesaver. It doesn't work under the Modern Memory Manager, but you can run your tests using the original 32-bit memory manager to get the benefit of this test.

MemErr checking

Despite admonitions from Apple, many programs don't check the value of MemErr , the low-memory global used to flag memory operation errors, after using memory-related Toolbox routines. When this test is active, QC will check MemErr after each Memory Manager trap.

Grow locked handle check

Inside Macintosh volume II has a warning: calling SetHandleSize() on a locked handle can fail if the block needs to be moved in order to grow. Unfortunately, MemErr is the only failure indication after such an occurrence. This test catches SetHandleSize() calls where the target handle is locked. [3]

Grow pointer checks

Nonrelocatable blocks, also known as pointer blocks, don't move when a heap shuffles, but they come with their own special limitations. When you try to grow a nonrelocatable block in the heap, if there's another nonrelocatable block "next" to it, the grow will fail. Like the "grow locked handle" test, this test will detect such failures and report them.

Stress Tests

Some applications behave well when they have plenty of RAM, but when forced to run in a smaller partition, they start to get flaky. Stress testing involves placing severe loads on the application by forcibly purging, scrambling, unloading, and otherwise molesting its heap. Well-written applications will be able to withstand such extreme conditions; other programs are bound to improve if their authors subject them to these tests.

Heap scramble

The state of an application's heap varies from run to run and from machine to machine. This variation makes it difficult to find heap-related bugs, like use of a stale handle or reliance on a handle's not moving. QC's heap scramble, like the standard Macsbug 'hs' command, moves all handles in the application heap each time a memory-moving trap call is made. This test is great for general-purpose stress testing, since it creates a condition not likely to occur during ordinary use.

Heap purge

Once a resource or segment has been marked as 'purgeable,' it's subject to being unloaded any time memory is allocated. If your code relies on the presence in memory of something which may have been purged, it can break when the Toolbox actually does a purge. The heap purge test causes QC to purge the heap after every memory-moving trap, meaning that anything marked purgeable will be flushed quickly.

Heap check

The heap check option tests the heap's structure for corruption at each memory-moving trap. By checking on every memory-moving trap, you're more likely to catch a heap-corrupting bug closer to the scene of the crime.

Invalidate free memory

This test puts a special salt value into all unallocated RAM within your application's partition. If your code attempts to dereference the contents of free memory, the salt value will cause a bus error. This check is extra-useful when combined with the heap purge test; as items are purged from the heap, the memory they used to occupy gets salted. If your code reuses it, QC will catch it in the act. Unfortunately, this test doesn't run under the Modern Memory Manager. Again, it runs fine on the PowerMac under the "classic" memory manager, so you can still use it for testing.

Eureka!

"Eureka" is Greek for "I have found it!" As mentioned above, the QC control panel will either beep or call _DebugStr when it encounters an error. The output string is very helpful in and of itself; it tells you what error was detected, what handles or parameters were involved, and the address of the last trap call in your code. QC only checks for errors after memory-related traps are called, so the "last trap called" field is useful since it can help you pinpoint the exact location of the error.

All you need to do is search backwards from the address in the last trap field to the previous trap call.

Some tests may cause bus errors, so instead of a nicely formatted QC message you get a plain debugger prompt. This most commonly occurs with the "dereference zero" and "write to zero" tests. Finding the precise location of these errors is up to you. QC uses a unique salt value for each of these tests, so you can tell which type of misbehavior caused the fault.

For the other cases where QC does a controlled stop because it saw something noteworthy, you end up in the debugger with the program counter (PC) set to an address within QC's address space. This poses a small problem for users of source-level debuggers. Since the source debugger doesn't issue the breakpoint, there's no way at break time to set a new breakpoint. To avoid this problem, you can use the API functions; if you write a small wrapper function which calls the QC routine of interest and breaks when an error occurs, you can break in the source-level debugger when the wrapper routine returns an error. Some debuggers also allow you to step out of the low-level debugger and back into your own function.

Using the API

The API is supplied as a C header file and three linkable 68K libraries, in MPW, Think, and CodeWarrior formats. The API contains calls to detect whether QC is active and/or installed, to activate or deactivate the current set of tests, to test the status of or activate/deactivate individual tests, and to instantly perform some types of tests.

Structurally the API is similar to most other Mac APIs; to use it, you use an enabling routine (QCActivate()) to start testing, then a series of get/set state calls to query, set, or clear various features and settings.

The API provides a great deal of flexibility, since you can easily turn tests on and off. By writing functions or macros to turn particular tests on and off, you can easily test at the function level. You can further configure your setup to run certain sets of tests at different points in the development cycle.

A minimal set of calls to use QC in your code might look like the code in Listing 1. Of course, in a real environment you should check error returns for all QC calls, since it's possible for the API routines to return a variety of error codes.

Listing 1: QCTools.c

Sample utility routines for using QC within applications. StartQCTesting turns on QC testing if QC is installed but not active. StopQCTesting turns QC testing off, and SetDevelopmentTests configures QC to use a particular set of tests.

QC tools

```
#include "QCAPI.h"
```

StartQCTesting

```
// turn on QC for testing
void StartQCTesting(void)
{
   if (QCInstalled())
      if (!QCIsActive())
         QCActivate(NULL);
}
```

StopQCTesting

```
// turn off QC when we're done
void StopQCTesting(void)
{
   if (QCInstalled())
      if (QCIsActive())
         QCDeactivate(NULL);
}
```

SetDevelopmentTests

```
// Turn on arbitrary set of tests depending on build type; we
// developed these test settings by guess and by gosh. You
// can define similar functions for other levels of testing.
void SetDevelopmentTests (void)
{
   QCErr testErr = noErr;
   testErr = QCSetTestState(qcValidateHandlePointers, TRUE);
   testErr = QCSetTestState(qcDetectWriteToZero, TRUE);
   testErr = QCSetTestState(qcDerefZeroCheck, TRUE);
   testErr = QCSetTestState(qcCheckDisposeRelease, TRUE);

   // make sure we see Macsbug error messages
   testErr = QCSetTestState(qcDebugBreaks, TRUE);
}
```

Callbacks

One really nifty feature of the QC API is that you can set custom callbacks that are invoked when QC detects an error. These callbacks can do extra testing or processing, depending on the error reported by QC. The QCInstallHandler() routine allows you to install a handler invoked each time an error occurs; within the callback routine, you can examine each error condition and you're interested in.

Listing 2: QCCallback.c

Sample callback which logs the error to a file instead of dropping into the debugger. This particular routine logs errors using the string that QC returns.

QC callbacks

```
#include "QCAPI.h"
```

ErrorToFileCallback

```
// the callback routine is defined as "typedef long (*QCCallBack)(QCPBPtr)";
// the QCParamPtr tells us what type of test generated the error and what
// error occurred.
long ErrorToFileCallback(QCPBPtr *theParam)
{
   QCErr anErr = kQCNoErr;
   StringPtr s = NULL;

   // get the error message that would normally go to Macsbug
   anErr = QCGetErrorText(theParam->errorID, s);
   if (anErr == kQCNoErr)
   {
      // log the message to our app log file
      LogEventMessage(kErrorEvt, s);
   }
   else
      LogEventMessage(kErrorEvent, 'QC internal error');
}
```

MyInstallQCCallBack

```
// call this routine to make QC start using the callback.
// In its original form, this routine gets called right before
// the start of the main event loop. The second parameter
// to QCInstallHandler is a refcon, so you can pass data to the handler
// routine when it gets called.
QCErr MyInstallQCCallBack (void)
{
  return QCInstallHandler(ErrorToFileCallBack, 0L);
}
```

MyUnInstallQCCallBack

```
// call this routine to make QC stop using the callback.
// In its original form, this routine gets called right before ExitToShell()
QCErr MyUnInstallQCCallBack (void)
{
  return QCRemoveHandler();
}
```

Documentation and Support

It's hard to effectively use a tool which has poor documentation or tech support. QC doesn't suffer from either. Like Metrowerks' CodeWarrior, QC comes packaged in a CD "jewel case" with no printed documentation. Instead, comprehensive DOCMaker-format documentation is included on the floppy; in addition to a "QuickStart" document, there's a full user's manual, plus a separate guide to the API. The disk also includes source code for "BadAPPL," a small application designed to provide a showcase for all the different types of errors QC can detect.

Onyx Tech offers technical support via e-mail; all my questions have been promptly, cheerfully, and accurately answered. Updaters for new versions are released to the Internet and several commercial online services; QC is presently at version 1.1.1.

Onyx has also taken the pleasant step of releasing a demo version of QC. This trend is finally catching on among development tools, and it's quite nice to be able to evaluate a tool in your own environment before laying down that hard-earned cash.

The Verdict

Since there are several free tools which incorporate various parts of QC's functionality, why would you pay $99? Three reasons: ease of use, completeness, and versatility. QC is easy to install and use, even for nontechnical users who might be doing QA or testing on your application, and it doesn't require any fluency in Macsbug. It incorporates a wider range of tests than any combination of other tools, and the ability to configure individual tests using the API is a terrific addition to an in-house development debugging suite.

Despite my enthusiasm, there are still a few things QC doesn't do. It doesn't find memory leaks; you'll still need the "leaks" dcmd for that. It doesn't give a precise cause for bus errors uncovered during the "dereference zero" and "write to zero" tests, so you'll have to find those on your own. Finally, QC could potentially lead to sloppy programming... after all, when you have such good tools for finding defects, your incentive not to make them in the first place may be reduced.

Onyx is planning leak detection, .SYM file support, and other user-requested improvements for their first post-PowerPC release, tentatively scheduled to ship in January or February of 1995.

For US$99, QC is a steal. Unless you're writing software that no one else ever uses, every defect you catch in house is one defect your freeware, shareware, or commercial customer will never see... and it's one defect you won't have to spend tech support or QA time finding and resolving. I give it two thumbs up.

Onyx Technology can be contacted via e-mail (OnyxTech@aol.com, D2238 on AppleLink, or 70550,1377 on CIS) or the more conventional routes: via phone at +1 813 795 7801, via fax at +1 813 792 5152, or via snail-mail at 7811 27th Avenue West; Bradenton, Florida, 34209 USA.

Bibliography and References

[1] McConnell, Steve. *Code Complete*, Microsoft Press, 1993; p. 625. I highly recommend this book for all software developers, especially the chapter on debugging.

[2] These tools are all available from ftp.apple.com in /pub/dts/mac.

[3] *Inside Macintosh* Vol. II, p. 34.

A Second Opinion

By Greg Marriott

I wrote *DoubleTrouble*, *EvenBetterBusError*, and *DisposeHandle* because I needed them and they didn't exist yet. Bo3b Johnson wrote *Blat*, *Leaks* and *ZapHandles* for pretty much the same reason – they didn't exist yet. If QC had existed when I was debugging application compatibility problems during 7.0 development, then I would have used it instead of taking the time to write my own tools.

If you don't buy this tool right now, then you're a bonehead. Don't come crying to me when your code sucks because you wouldn't spend $100.

To receive information on any products advertised in this issue, send your request via Internet: productinfo@xplain.com

By Scott T Boyd, Editor

The Very First Issue

Ten years doesn't seem like so long ago

I don't think I could forget the first time I saw MacTutor. Or was it MacTech? Ok, so I forgot that much – but the rest I won't forget. My friend Roger had somehow come across a copy of a magazine about how to program the Macintosh. He brought it to one of our programmer get-togethers. Joy! It was pretty hard to get any documentation, much less good documentation, about programming the Macintosh at Texas A&M back in early 1985.

Now, to understand Roger, it might help to know a little about him. I bought my first Mac brand new for $2500. That got me a 128K Mac and an ImageWriter. I gave Roger a demo. He spent an hour or so almost silently watching me give him a demo. I got the distinct impression that he wasn't impressed at all. Boy, was I wrong! The next day he went out and bought as complete a Macintosh system as you could buy. His excitement rubbed off on me, and I went back and bought a second floppy drive. I still remember shopping to buy floppies (800K). Those things were expensive, something like $10 each. It makes me laugh remembering this because now I consider floppies a nuisance. They just clutter up my shelves.

At any rate, Roger's infectious excitement isn't the kind to stay bottled up, and he came to our meeting with a couple of copies of this new magazine, one copy for me, and one for Greg Marriott. Now, these had to be the worst copies I had ever seen. Somehow the toner had gotten only partially fixed onto the paper. We had to handle the pages very carefully, but we didn't mind a bit. With this one little magazine, we went from having next to nothing in the way of practical programmer's advice, experience, and examples, to having a rich monthly resource! Given that Roger was the only one with money to spend on such things, he made copies of the first three issue for us before we managed to get our own subscriptions. I still have those bad xerox copies stored carefully as the treasures that they are.

On the next few pages we have reproduced the entire first issue. We scanned the only real copy that we had left. Since I work in the remote wilds of Montara, California, I didn't see the original before I saw the scans. I thought something must have gone wrong in the scanning process – everything was far muddier than I expected. I had remembered that the first few issues were done on an ImageWriter, and probably at 72 dpi, but I hadn't remembered that the magazine itself looked like it had been reproduced on a cheap copier. On closer inspection, it's clear now that much of the magazine was also put together using good old cut and paste techniques. This serves as a reminder of how far we've come in the production of the magazine. Rather than 72dpi, we image at 2540dpi. Rather than use only MacWrite, MacPaint, and MacDraw, we use QuarkXPress, PhotoShop, Illustrator, and a host of other publishing and imaging utilities. There's no cheap copier involved in today's production; we print on a Web press and do a lot of four-color printing.

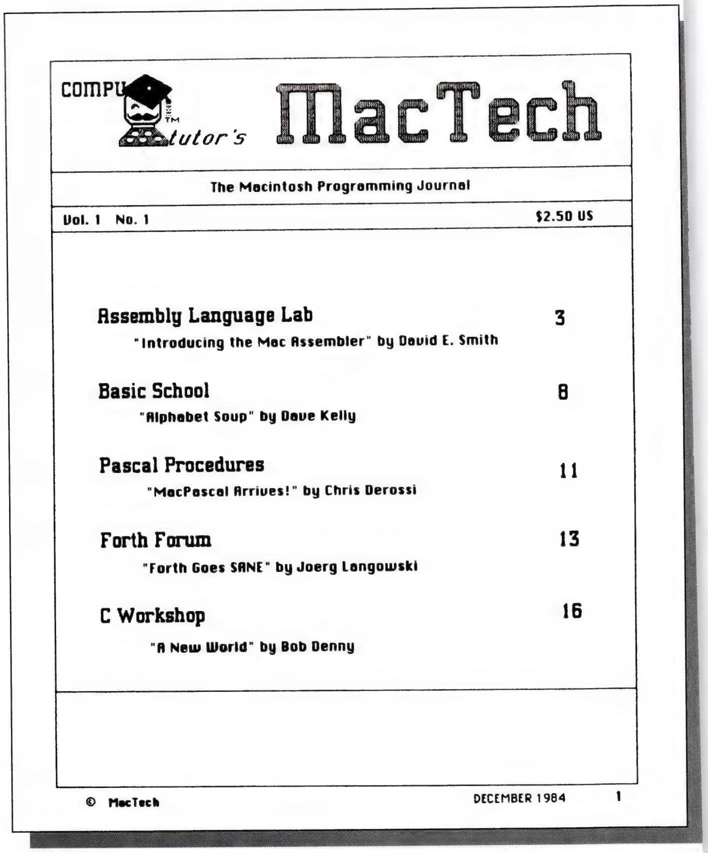
We discovered something interesting while putting this

Scott T Boyd – Scott first wrote for MacTutor in the September 1986 issue on the topic of *The Pop-up, Two Dimensional, Random Access, Scroll Bar Menu*. Although he credits MacTutor with helping to launch his career, he would like to point out that this particular concept didn't make him filthy rich (as kindly requested in one of his articles). Perhaps that has something to do with why he continues to work, currently as Editor of MacTech Magazine and Proprietor of The MacHax™ Group.

article together. I passed around some different scans to a few folks to see which settings they thought produced the best results. Steve Kiene went and compared them to his copy of the first issue. He noticed something right away. His magazine was entitled "MacTutor™ (Formerly MacTech)" while our copy read "COMPUtutor's MacTech". Both were clearly labeled Vol. 1 No. 1. There's a story in there somewhere about a company called Machine Technologies, but we'll leave that for someone else to tell. One thing has definitely changed during the past decade. Our masthead now reads "MacTech Magazine (Formerly MacTutor)", bringing us full circle.

The first issue was twenty pages, and carried only a couple of ads, all for the magazine itself. By the following December, it had grown to seventy two pages, and was carried in stores in twenty nine states as well as West Germany, Japan, and Sweden. In addition, thirty two advertisers came on board in that first year. Some of those names you'll see are still with us as regular advertisers (e.g. MacNosy and Mainstay). Many are still writing and selling Macintosh software, including Alsoft, Capilano Computing, and FWB. Some have faded into fond memories.

The first issue rang in the beginning of the era of programming on the Macintosh for the Macintosh. Prior to this, just about all Mac programming had been done on the Lisa. We've watched such an event pass before us again this year. Last year, about the only way to program a Power Macintosh was on an IBM unix box. Power Mac programmers rang in the new year with Code Warrior, a programming environment for the Power Macintosh *on* the Macintosh (Power and 68K).

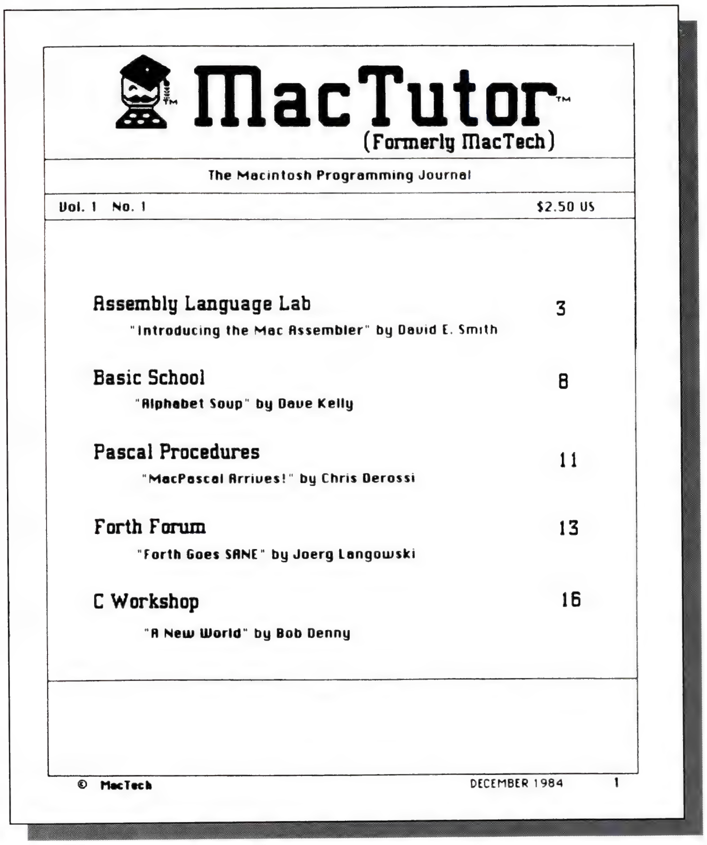

COMPUtutor's MacTech

The Macintosh Programming Journal

Vol. 1 No. 1 $2.50 US

Assembly Language Lab "Introducing the Mac Assembler" by David E. Smith	3
Basic School "Alphabet Soup" by Dave Kelly	8
Pascal Procedures "MacPascal Arrives!" by Chris Derossi	11
Forth Forum "Forth Goes SANE" by Joerg Langowski	13
C Workshop "A New World" by Bob Denny	16

© MacTech DECEMBER 1984 1


MacTutor™ (Formerly MacTech)

The Macintosh Programming Journal

Vol. 1 No. 1 $2.50 US

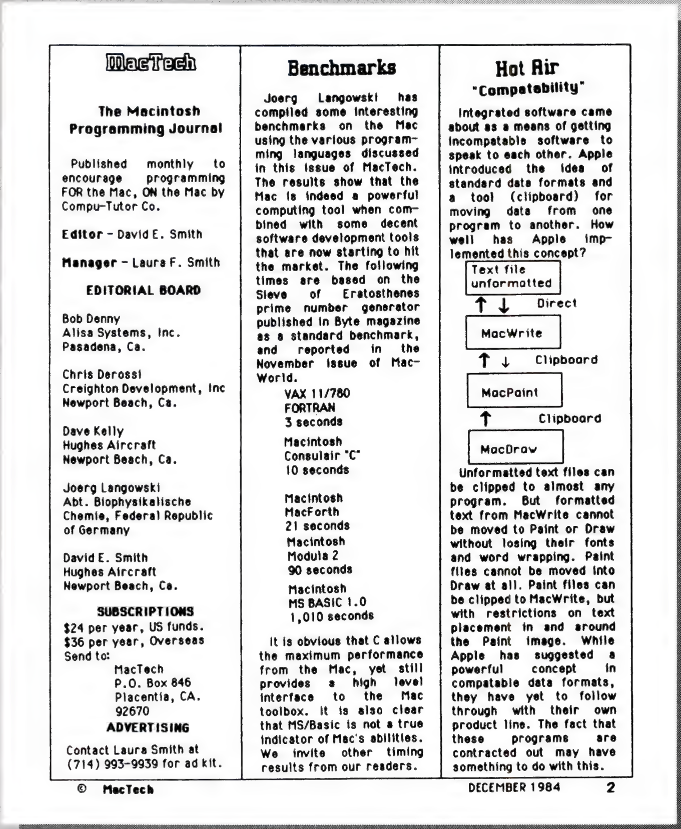
Assembly Language Lab "Introducing the Mac Assembler" by David E. Smith	3
Basic School "Alphabet Soup" by Dave Kelly	8
Pascal Procedures "MacPascal Arrives!" by Chris Derossi	11
Forth Forum "Forth Goes SANE" by Joerg Langowski	13
C Workshop "A New World" by Bob Denny	16

© MacTech DECEMBER 1984 1

We're not using 128K Macs with a single (singing) floppy and no hard drive any more, although there are undoubtedly those among us who still scrape and save to buy those necessary supplies. The complexity and sophistication of most of our tools have increased dramatically, but here we are, still programming and learning about a machine we call Macintosh.

Many things have changed in the realm of Macintosh programming over the past decade, but one thing has not. See page 20 of the first issue. "A no-nonsense, no fluff Journal devoted to software development FOR Mac, ON Mac. Let MacTech's editorial board teach you the Macintosh technology of windows, quickdraw, events and resources. We have assembled a team of professionals to uncover and explain Mac's secrets." The programming staff has changed somewhat, but still includes professionals, some of whom are busy building the next generation of Macintosh.

We hope you enjoy reading (or rereading!) this piece of ancient history as we celebrate MacTech Magazine's Tenth Anniversary. In this business, a decade is nearly a lifetime. After all this time and almost one hundred and twenty issues of the magazine later, we're pleased to continue to bring you the latest and greatest programming information for and about the Macintosh, on the Macintosh.



MacTech

The Macintosh Programming Journal

Published monthly to encourage programming FOR the Mac, ON the Mac by Compu-Tutor Co.

Editor - David E. Smith

Manager - Laura F. Smith

SUBSCRIPTIONS
$24 per year, US funds.
$36 per year, Overseas
Send to:
MacTech
P.O. Box 846
Placentia, CA.
92670

ADVERTISING

Contact Laura Smith at (714) 993-9939 for ad kit.

Benchmarks

Joerg Langowski has compiled some interesting benchmarks on the Mac using the various programming languages discussed in this issue of MacTech. The results show that the Mac is indeed a powerful computing tool when combined with some decent software development tools that are now starting to hit the market. The following times are based on the Sieve of Eratosthenes prime number generator published in Byte magazine as a standard benchmark, and reported in the November issue of MacWorld.

VAX 11/780
FORTRAN
3 seconds

Macintosh
Consulair "C"
10 seconds

Macintosh
MacForth
21 seconds

Macintosh
Modula 2
90 seconds

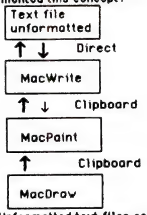
Macintosh
MS BASIC 1.0
1,010 seconds

It is obvious that C allows the maximum performance from the Mac, yet still provides a high level interface to the Mac toolbox. It is also clear that MS/Basic is not a true indicator of Mac's abilities. We invite other timing results from our readers.

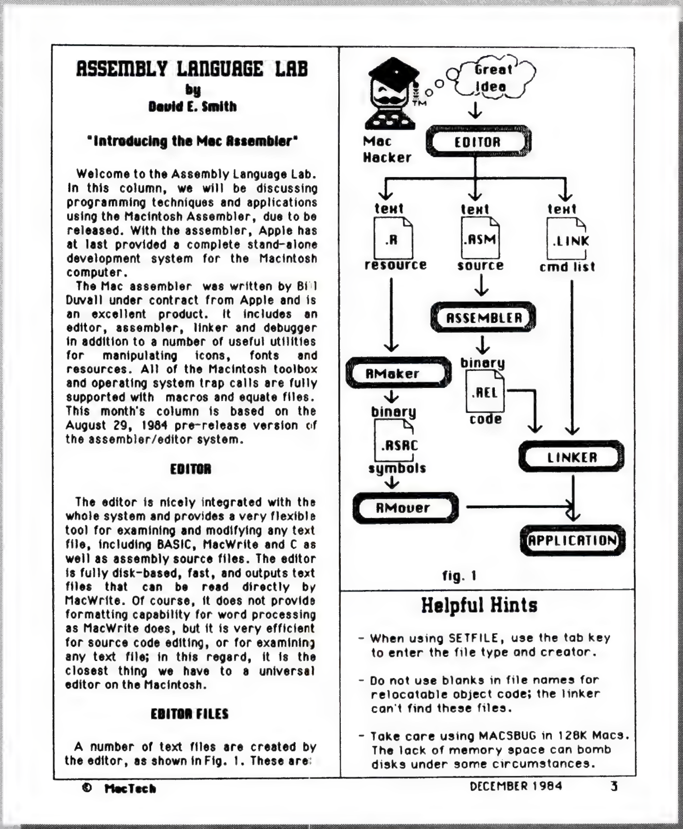
Hot Air

"Compatability"

Integrated software came about as a means of getting incompatable software to speak to each other. Apple introduced the idea of standard data formats and a tool (clipboard) for moving data from one program to another. How well has Apple implemented this concept?



Unformatted text files can be clipped to almost any program. But formatted text from MacWrite cannot be moved to Paint or Draw without losing their fonts and word wrapping. Paint files cannot be moved into Draw at all. Paint files can be clipped to MacWrite, but with restrictions on text placement in and around the Paint image. While Apple has suggested a powerful concept in compatable data formats, they have yet to follow through with their own product line. The fact that these programs are contracted out may have something to do with this.



ASSEMBLY LANGUAGE LAB
by
David E. Smith

"Introducing the Mac Assembler"

Welcome to the Assembly Language Lab. In this column, we will be discussing programming techniques and applications using the Macintosh Assembler, due to be released. With the assembler, Apple has at last provided a complete stand-alone development system for the Macintosh computer.

The Mac assembler was written by Bill Duvall under contract from Apple and is an excellent product. It includes an editor, assembler, linker and debugger in addition to a number of useful utilities for manipulating icons, fonts and resources. All of the Macintosh toolbox and operating system trap calls are fully supported with macros and equate files. This month's column is based on the August 29, 1984 pre-release version of the assembler/editor system.

EDITOR

The editor is nicely integrated with the whole system and provides a very flexible tool for examining and modifying any text file, including BASIC, MacWrite and C as well as assembly source files. The editor is fully disk-based, fast, and outputs text files that can be read directly by MacWrite. Of course, it does not provide formatting capability for word processing as MacWrite does, but it is very efficient for source code editing, or for examining any text file; in this regard, it is the closest thing we have to a universal editor on the Macintosh.

EDITOR FILES

A number of text files are created by the editor, as shown in Fig. 1. These are:

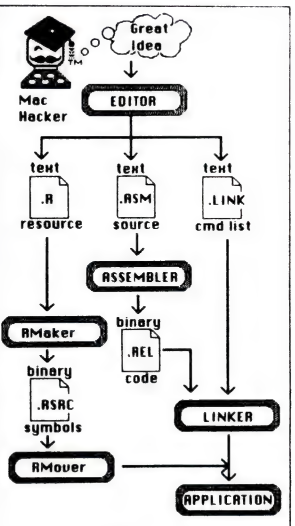
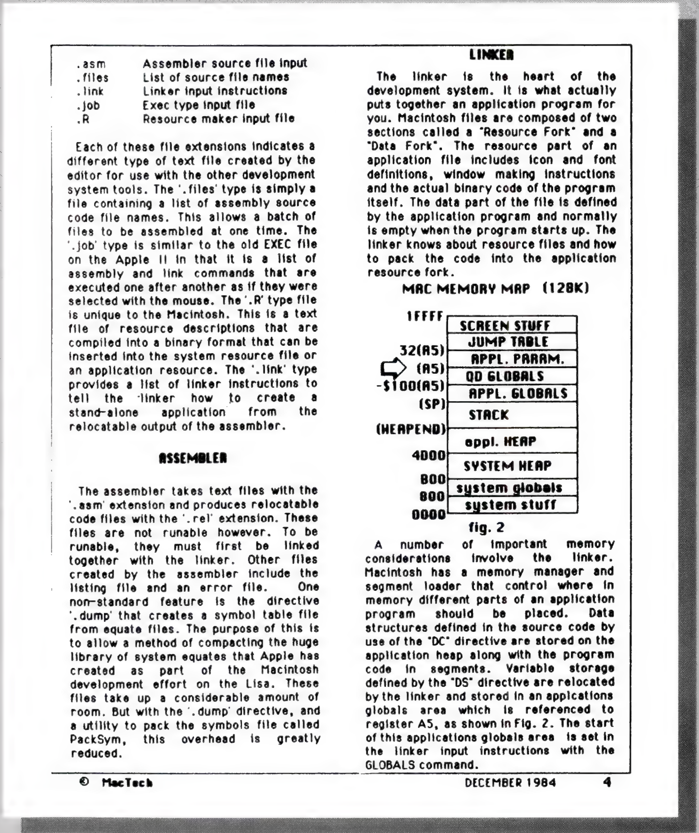


fig. 1

Helpful Hints

- When using SETFILE, use the tab key to enter the file type and creator.
- Do not use blanks in file names for relocatable object code; the linker can't find these files.
- Take care using MACSBUG in 128K Macs. The lack of memory space can bomb disks under some circumstances.



.asm	Assembler source file input
.files	List of source file names
.link	Linker input instructions
.job	Exec type input file
.R	Resource maker input file

Each of these file extensions indicates a different type of text file created by the editor for use with the other development system tools. The '.files' type is simply a file containing a list of assembly source code file names. This allows a batch of files to be assembled at one time. The '.job' type is similar to the old EXEC file on the Apple II in that it is a list of assembly and link commands that are executed one after another as if they were selected with the mouse. The '.R' type file is unique to the Macintosh. This is a text file of resource descriptions that are compiled into a binary format that can be inserted into the system resource file or an application resource. The '.link' type provides a list of linker instructions to tell the linker how to create a stand-alone application from the relocatable output of the assembler.

ASSEMBLER

The assembler takes text files with the '.asm' extension and produces relocatable code files with the '.rel' extension. These files are not runable however. To be runable, they must first be linked together with the linker. Other files created by the assembler include the listing file and an error file. One non-standard feature is the directive '.dump' that creates a symbol table file from equate files. The purpose of this is to allow a method of compacting the huge library of system equates that Apple has created as part of the Macintosh development effort on the Lisa. These files take up a considerable amount of room. But with the '.dump' directive, and a utility to pack the symbols file called PackSym, this overhead is greatly reduced.

LINKER

The linker is the heart of the development system. It is what actually puts together an application program for you. Macintosh files are composed of two sections called a "Resource Fork" and a "Data Fork". The resource part of an application file includes icon and font definitions, window making instructions and the actual binary code of the program itself. The data part of the file is defined by the application program and normally is empty when the program starts up. The linker knows about resource files and how to pack the code into the application resource fork.

MAC MEMORY MAP (128K)

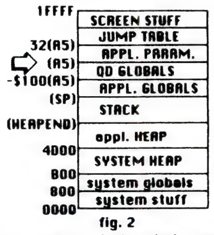
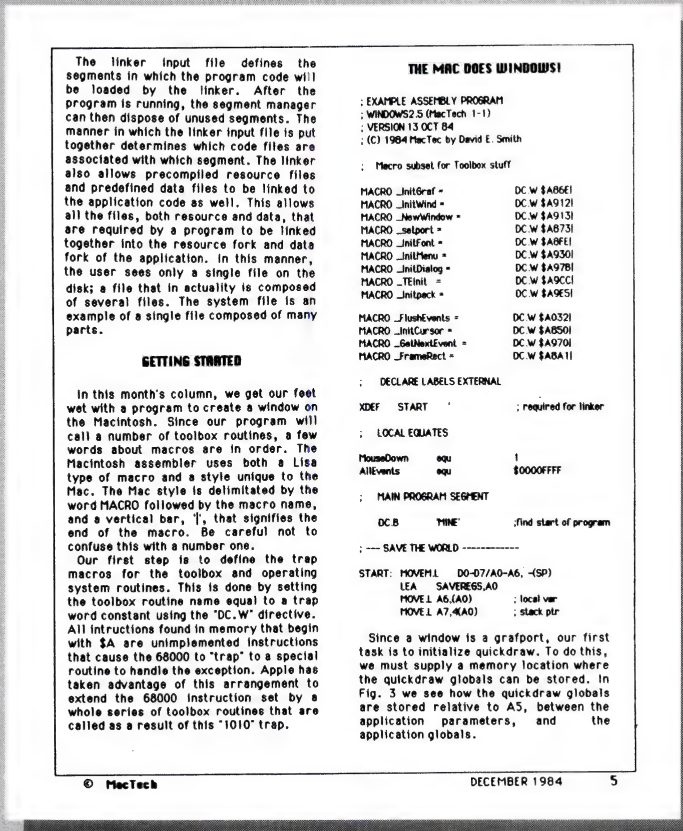


fig. 2

A number of important memory considerations involve the linker. Macintosh has a memory manager and segment loader that control where in memory different parts of an application program should be placed. Data structures defined in the source code by use of the "DC" directive are stored on the application heap along with the program code in segments. Variable storage defined by the "DS" directive are relocated by the linker and stored in an applcations globals area which is referenced to register A5, as shown in Fig. 2. The start of this applications globals area is set in the linker input instructions with the GLOBALS command.



The linker input file defines the segments in which the program code will be loaded by the linker. After the program is running, the segment manager can then dispose of unused segments. The manner in which the linker input file is put together determines which code files are associated with which segment. The linker also allows precompiled resource files and predefined data files to be linked to the application code as well. This allows all the files, both resource and data, that are required by a program to be linked together into the resource fork and data fork of the application. In this manner, the user sees only a single file on the disk; a file that in actuality is composed of several files. The system file is an example of a single file composed of many parts.

GETTING STARTED

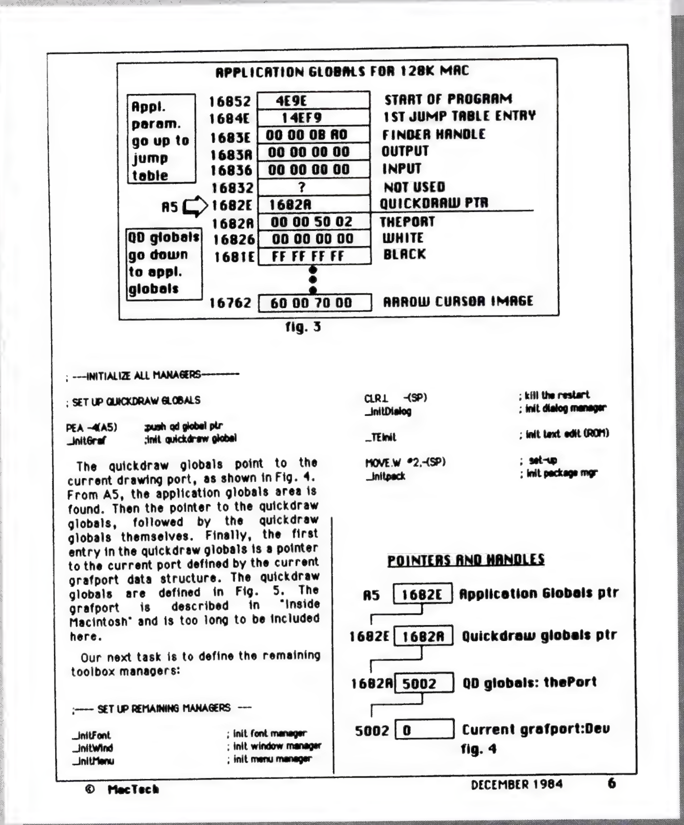
In this month's column, we get our feet wet with a program to create a window on the Macintosh. Since our program will call a number of toolbox routines, a few words about macros are in order. The Macintosh assembler uses both a Lisa type of macro and a style unique to the Mac. The Mac style is delimitated by the word MACRO followed by the macro name, and a vertical bar, '|', that signifies the end of the macro. Be careful not to confuse this with a number one.

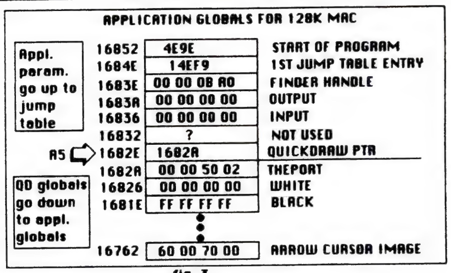
Our first step is to define the trap macros for the toolbox and operating system routines. This is done by setting the toolbox routine name equal to a trap word constant using the "DC.W" directive. All intructions found in memory that begin with $A are unimplemented instructions that cause the 68000 to "trap" to a special routine to handle the exception. Apple has taken advantage of this arrangement to extend the 68000 instruction set by a whole series of toolbox routines that are called as a result of this "1010" trap.

THE MAC DOES WINDOWS!

```
; EXAMPLE ASSEMBLY PROGRAM
; WINDOWS2.5 (MacTech 1-1)
; VERSION 13 OCT 84
; (C) 1984 MacTec by David E. Smith

;   Macro subset for Toolbox stuff

MACRO _InitGraf =           DC.W $A86E|
MACRO _InitWind =           DC.W $A912|
MACRO _NewWindow =          DC.W $A913|
MACRO _setport =            DC.W $A873|
MACRO _InitFont =           DC.W $A8FE|
MACRO _InitMenu =           DC.W $A930|
MACRO _InitDialog =         DC.W $A978|
MACRO _TEInit   =           DC.W $A9CC|
MACRO _Initpack =           DC.W $A9E5|

MACRO _FlushEvents =        DC.W $A032|
MACRO _InitCursor =         DC.W $A850|
MACRO _GetNextEvent =       DC.W $A970|
MACRO _FrameRect =          DC.W $A8A1|

;    DECLARE LABELS EXTERNAL

XDEF    START                   ; required for linker

;    LOCAL EQUATES

MouseDown       equ         1
AllEvents       equ         $0000FFFF

;    MAIN PROGRAM SEGMENT

    DC.B        'MINE'          ;find start of program

; --- SAVE THE WORLD ------------

START:  MOVEM.L   D0-D7/A0-A6, -(SP)
        LEA       SAVEREGS,A0
        MOVE.L    A6,(A0)       ; local var
        MOVE.L    A7,4(A0)      ; stack ptr
```

Since a window is a grafport, our first task is to initialize quickdraw. To do this, we must supply a memory location where the quickdraw globals can be stored. In Fig. 3 we see how the quickdraw globals are stored relative to A5, between the application parameters, and the application globals.





APPLICATION GLOBALS FOR 128K MAC

Appl. param. go up to jump table	16852	4E9E	START OF PROGRAM
	1684E	14EF9	1ST JUMP TABLE ENTRY
	1683E	00 00 0B A0	FINDER HANDLE
	1683A	00 00 00 00	OUTPUT
	16836	00 00 00 00	INPUT
	16832	?	NOT USED
A5 ⇨	1682E	1682A	QUICKDRAW PTR
QD globals go down to appl. globals	1682A	00 00 50 02	THEPORT
	16826	00 00 00 00	WHITE
	1681E	FF FF FF FF	BLACK
	⋮		
	16762	60 00 70 00	ARROW CURSOR IMAGE

fig. 3

```
; ---INITIALIZE ALL MANAGERS---------

; SET UP QUICKDRAW GLOBALS

PEA -4(A5)        ;push qd global ptr
_InitGraf         ;init quickdraw global
```

The quickdraw globals point to the current drawing port, as shown in Fig. 4. From A5, the application globals area is found. Then the pointer to the quickdraw globals, followed by the quickdraw globals themselves. Finally, the first entry in the quickdraw globals is a pointer to the current port defined by the current grafport data structure. The quickdraw globals are defined in Fig. 5. The grafport is described in "Inside Macintosh" and is too long to be included here.

Our next task is to define the remaining toolbox managers:

```
;---- SET UP REMAINING MANAGERS ---

_InitFont                   ; init font manager
_InitWind                   ; init window manager
_InitMenu                   ; init menu manager
```

```
CLR.L   -(SP)               ; kill the restart
_InitDialog                 ; init dialog manager

_TEInit                     ; init text edit (ROM)

MOVE.W  #2,-(SP)            ;  set-up
_Initpack                   ; init package mgr
```

POINTERS AND HANDLES

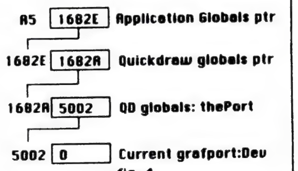
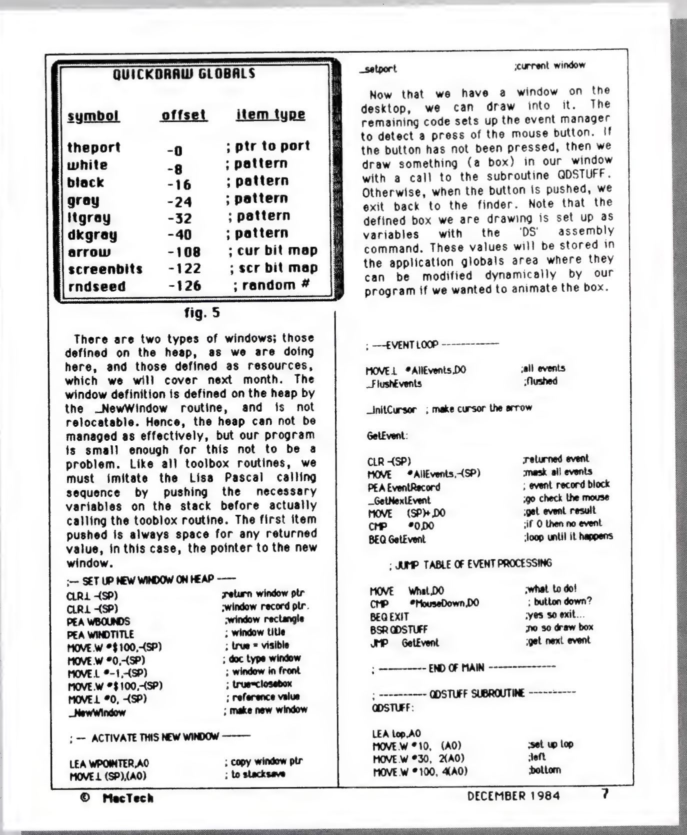


fig. 4



QUICKDRAW GLOBALS

symbol	offset	item type
theport	-0	; ptr to port
white	-8	; pattern
black	-16	; pattern
gray	-24	; pattern
ltgray	-32	; pattern
dkgray	-40	; pattern
arrow	-108	; cur bit map
screenbits	-122	; scr bit map
rndseed	-126	; random #

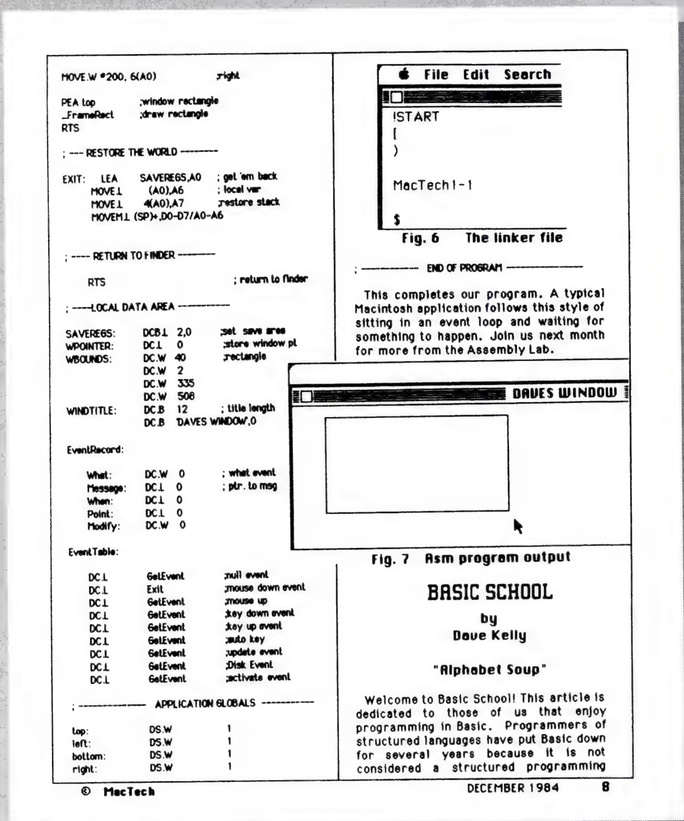
fig. 5

There are two types of windows; those defined on the heap, as we are doing here, and those defined as resources, which we will cover next month. The window definition is defined on the heap by the _NewWindow routine, and is not relocatable. Hence, the heap can not be managed as effectively, but our program is small enough for this not to be a problem. Like all toolbox routines, we must imitate the Lisa Pascal calling sequence by pushing the necessary variables on the stack before actually calling the tooblox routine. The first item pushed is always space for any returned value, in this case, the pointer to the new window.

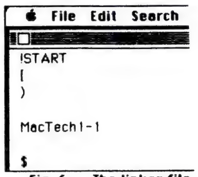
```
;-- SET UP NEW WINDOW ON HEAP ----
CLR.L -(SP)                 ;return window ptr
CLR.L -(SP)                 ;window record ptr.
PEA WBOUNDS                 ;window rectangle
PEA WINDTITLE               ; window title
MOVE.W #$100,-(SP)          ; true = visible
MOVE.W #0,-(SP)             ; doc type window
MOVE.L #-1,-(SP)            ; window in front
MOVE.W #$100,-(SP)          ; true=closebox
MOVE.L #0, -(SP)            ; reference value
_NewWindow                  ; make new window

; -- ACTIVATE THIS NEW WINDOW -------

LEA WPOINTER,A0             ; copy window ptr
MOVE.L (SP),(A0)            ; to stacksave
_setport                    ;current window
```

Now that we have a window on the desktop, we can draw into it. The remaining code sets up the event manager to detect a press of the mouse button. If the button has not been pressed, then we draw something (a box) in our window with a call to the subroutine QDSTUFF. Otherwise, when the button is pushed, we exit back to the finder. Note that the defined box we are drawing is set up as variables with the 'DS' assembly command. These values will be stored in the application globals area where they can be modified dynamically by our program if we wanted to animate the box.

```
; ---EVENT LOOP ------------

MOVE.L  #AllEvents,D0       ;all events
_FlushEvents                ;flushed

_InitCursor  ; make cursor the arrow

GetEvent:

CLR -(SP)                   ;returned event
MOVE    #AllEvents,-(SP)    ;mask all events
PEA EventRecord             ; event record block
_GetNextEvent               ;go check the mouse
MOVE    (SP)+,D0            ;get event result
CMP     #0,D0               ;if 0 then no event
BEQ GetEvent                ;loop until it happens

    ; JUMP TABLE OF EVENT PROCESSING

MOVE    What,D0             ;what to do!
CMP     #MouseDown,D0       ; button down?
BEQ EXIT                    ;yes so exit...
BSR QDSTUFF                 ;no so draw box
JMP     GetEvent            ;get next event

; ---------- END OF MAIN --------------

; ---------- QDSTUFF SUBROUTINE ----------
QDSTUFF:

LEA top,A0
MOVE.W #10,  (A0)           ;set up top
MOVE.W #30,  2(A0)          ;left
MOVE.W #100, 4(A0)          ;bottom
```



```
MOVE.W #200, 6(A0)              ;right

PEA top           ;window rectangle
_FrameRect        ;draw rectangle
RTS

; --- RESTORE THE WORLD ---------

EXIT:   LEA      SAVEREGS,A0    ; get 'em back
      MOVE.L     (A0),A6        ; local var
      MOVE.L     4(A0),A7       ;restore stack
      MOVEM.L (SP)+,D0-D7/A0-A6


; ---- RETURN TO FINDER --------

    RTS                               ; return to finder

; ----LOCAL DATA AREA -----------

SAVEREGS:      DCB.L  2,0       ;set  save area
WPOINTER:      DC.L    0        ;store window pt
WBOUNDS:       DC.W   40        ;rectangle
               DC.W   2
               DC.W   335
               DC.W   508
WINDTITLE:     DC.B   12        ; title length
               DC.B   'DAVES WINDOW',0

EventRecord:

    What:      DC.W   0         ; what event
    Message:   DC.L   0         ; ptr. to msg
    When:      DC.L   0
    Point:     DC.L   0
    Modify:    DC.W   0

EventTable:

    DC.L       GetEvent         ;null event
    DC.L       Exit             ;mouse down event
    DC.L       GetEvent         ;mouse up
    DC.L       GetEvent         ;key down event
    DC.L       GetEvent         ;key up event
    DC.L       GetEvent         ;auto key
    DC.L       GetEvent         ;update event
    DC.L       GetEvent         ;Disk Event
    DC.L       GetEvent         ;activate event

; ------------- APPLICATION GLOBALS ----------

top:           DS.W          1
left:          DS.W          1
bottom:        DS.W          1
right:         DS.W          1
```



```
!START
[
)

MacTech1-1

$
```

Fig. 6 The linker file

```
; ----------- END OF PROGRAM --------------
```

This completes our program. A typical Macintosh application follows this style of sitting in an event loop and waiting for something to happen. Join us next month for more from the Assembly Lab.

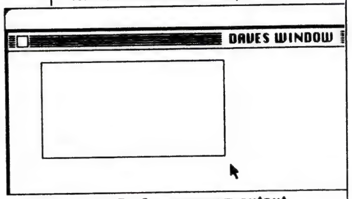


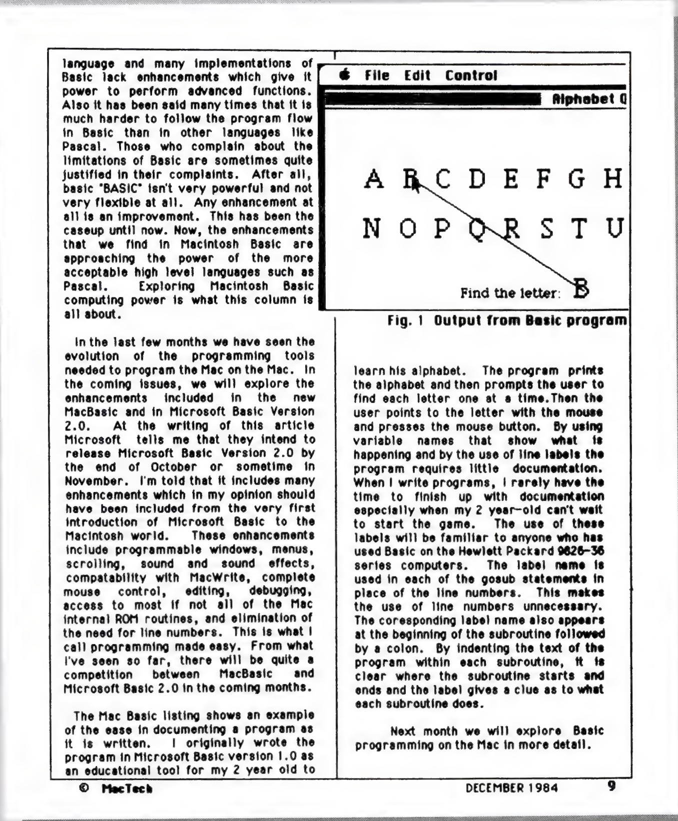
Fig. 7 Asm program output

BASIC SCHOOL

by
Dave Kelly

"Alphabet Soup"

Welcome to Basic School! This article is dedicated to those of us that enjoy programming in Basic. Programmers of structured languages have put Basic down for several years because it is not considered a structured programming



language and many implementations of Basic lack enhancements which give it power to perform advanced functions. Also it has been said many times that it is much harder to follow the program flow in Basic than in other languages like Pascal. Those who complain about the limitations of Basic are sometimes quite justified in their complaints. After all, basic "BASIC" isn't very powerful and not very flexible at all. Any enhancement at all is an improvement. This has been the caseup until now. Now, the enhancements that we find in Macintosh Basic are approaching the power of the more acceptable high level languages such as Pascal. Exploring Macintosh Basic computing power is what this column is all about.

In the last few months we have seen the evolution of the programming tools needed to program the Mac on the Mac. In the coming issues, we will explore the enhancements included in the new MacBasic and in Microsoft Basic Version 2.0. At the writing of this article Microsoft tells me that they intend to release Microsoft Basic Version 2.0 by the end of October or sometime in November. I'm told that it includes many enhancements which in my opinion should have been included from the very first introduction of Microsoft Basic to the Macintosh world. These enhancements include programmable windows, menus, scrolling, sound and sound effects, compatability with MacWrite, complete mouse control, editing, debugging, access to most if not all of the Mac internal ROM routines, and elimination of the need for line numbers. This is what I call programming made easy. From what I've seen so far, there will be quite a competition between MacBasic and Microsoft Basic 2.0 in the coming months.

The Mac Basic listing shows an example of the ease in documenting a program as it is written. I originally wrote the program in Microsoft Basic version 1.0 as an educational tool for my 2 year old to

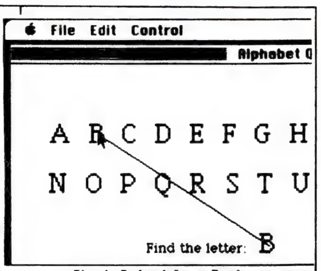
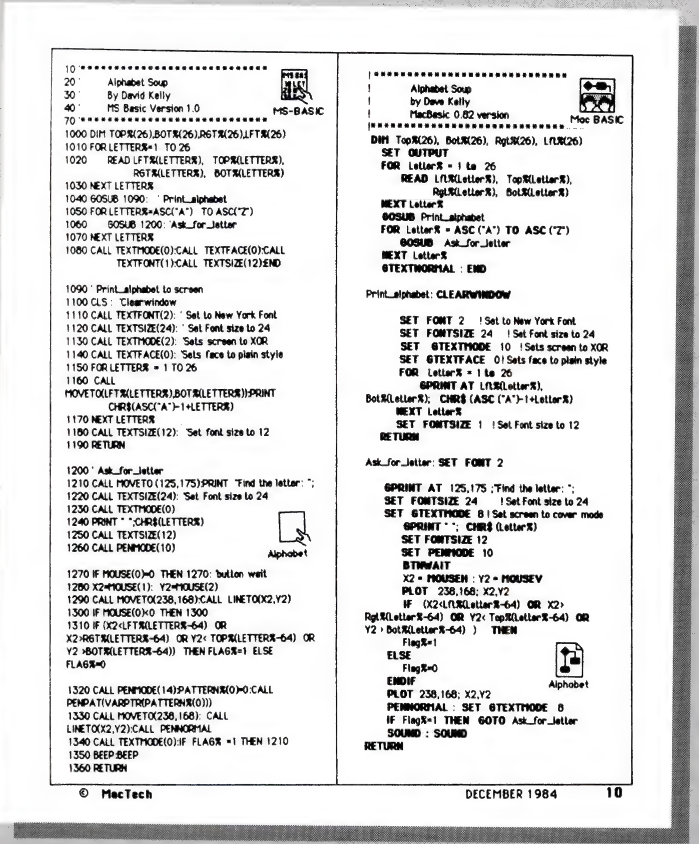


Fig. 1 Output from Basic program

learn his alphabet. The program prints the alphabet and then prompts the user to find each letter one at a time.Then the user points to the letter with the mouse and presses the mouse button. By using variable names that show what is happening and by the use of line labels the program requires little documentation. When I write programs, I rarely have the time to finish up with documentation especially when my 2 year-old can't wait to start the game. The use of these labels will be familiar to anyone who has used Basic on the Hewlett Packard 9826-36 series computers. The label name is used in each of the gosub statements in place of the line numbers. This makes the use of line numbers unnecessary. The coresponding label name also appears at the beginning of the subroutine followed by a colon. By indenting the text of the program within each subroutine, it is clear where the subroutine starts and ends and the label gives a clue as to what each subroutine does.

Next month we will explore Basic programming on the Mac in more detail.



```
10 '*****************************
20 '        Alphabet Soup
30 '        By David Kelly
40 '        MS Basic Version 1.0
70 '*****************************
1000 DIM TOP%(26),BOT%(26),RGT%(26),LFT%(26)
1010 FOR LETTER%=1  TO 26
1020      READ LFT%(LETTER%),   TOP%(LETTER%),
               RGT%(LETTER%),   BOT%(LETTER%)
1030 NEXT LETTER%
1040 GOSUB 1090:   ' Print_alphabet
1050 FOR LETTER%=ASC("A")   TO ASC("Z")
1060      GOSUB 1200: 'Ask_for_letter
1070 NEXT LETTER%
1080 CALL TEXTMODE(0):CALL  TEXTFACE(0):CALL
          TEXTFONT(1):CALL  TEXTSIZE(12):END

1090 ' Print_alphabet to screen
1100 CLS :  'Clearwindow
1110 CALL TEXTFONT(2):  ' Set to New York Font
1120 CALL TEXTSIZE(24):  ' Set Font size to 24
1130 CALL TEXTMODE(2):  'Sets screen to XOR
1140 CALL TEXTFACE(0):  'Sets face to plain style
1150 FOR LETTER% = 1 TO 26
1160  CALL
MOVETO(LFT%(LETTER%),BOT%(LETTER%)):PRINT
        CHR$(ASC("A")-1+LETTER%)
1170 NEXT LETTER%
1180 CALL TEXTSIZE(12):   'Set font size to 12
1190 RETURN

1200 ' Ask_for_letter
1210 CALL MOVETO (125,175):PRINT  "Find the letter: ";
1220 CALL TEXTSIZE(24):  'Set Font size to 24
1230 CALL TEXTMODE(0)
1240 PRINT " ";CHR$(LETTER%)
1250 CALL TEXTSIZE(12)
1260 CALL PENMODE(10)

1270 IF MOUSE(0)=0  THEN 1270:  'button wait
1280 X2=MOUSE(1):  Y2=MOUSE(2)
1290 CALL MOVETO(238,168):CALL  LINETO(X2,Y2)
1300 IF MOUSE(0)<0  THEN 1300
1310 IF (X2<LFT%(LETTER%-64)   OR
X2>RGT%(LETTER%-64)   OR Y2< TOP%(LETTER%-64)   OR
Y2 >BOT%(LETTER%-64))   THEN FLAG%=1  ELSE
FLAG%=0

1320 CALL PENMODE(14):PATTERN%(0)=0:CALL
PENPAT(VARPTR(PATTERN%(0)))
1330 CALL MOVETO(238,168):  CALL
LINETO(X2,Y2):CALL  PENNORMAL
1340 CALL TEXTMODE(0):IF  FLAG% =1 THEN 1210
1350 BEEP:BEEP
1360 RETURN
```

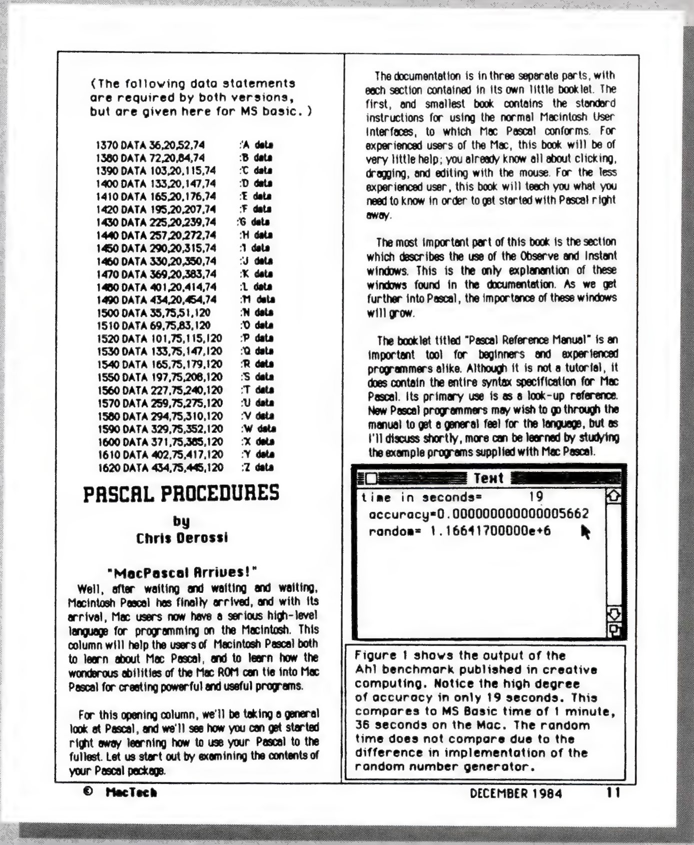
MS-BASIC

```
!*****************************
!        Alphabet Soup
!        by Dave Kelly
!        MacBasic 0.82 version
!*****************************
DIM  Top%(26),  Bot%(26),  Rgt%(26),  Lft%(26)
  SET  OUTPUT
  FOR  Letter% = 1 to  26
      READ  Lft%(Letter%),   Top%(Letter%),
            Rgt%(Letter%),   Bot%(Letter%)
  NEXT Letter%
  GOSUB  Print_alphabet
  FOR  Letter% = ASC ("A")  TO  ASC ("Z")
      GOSUB   Ask_for_letter
  NEXT  Letter%
  GTEXTNORMAL  :  END

Print_alphabet:  CLEARWINDOW

      SET  FONT  2     ! Set to New York Font
      SET  FONTSIZE  24     ! Set Font size to 24
      SET   GTEXTMODE   10   ! Sets screen to XOR
      SET  GTEXTFACE   0! Sets face to plain style
      FOR  Letter% = 1 to  26
          GPRINT AT  Lft%(Letter%),
Bot%(Letter%);   CHR$ (ASC ("A")-1+Letter%)
      NEXT  Letter%
      SET  FONTSIZE  1   ! Set Font size to 12
  RETURN

Ask_for_letter:  SET  FONT  2

    GPRINT  AT  125,175 ;"Find the letter: ";
    SET  FONTSIZE  24       ! Set Font size to 24
    SET  GTEXTMODE  8 ! Set screen to cover mode
       GPRINT " ";  CHR$ (Letter%)
       SET FONTSIZE 12
       SET  PENMODE  10
       BTNWAIT
       X2 = MOUSEH  :  Y2 = MOUSEV
       PLOT   238,168;  X2,Y2
       IF   (X2<Lft%(Letter%-64)  OR  X2>
Rgt%(Letter%-64)  OR  Y2< Top%(Letter%-64)  OR
Y2 > Bot%(Letter%-64)  )    THEN
       Flag%=1
    ELSE
       Flag%=0
    ENDIF
    PLOT  238,168;  X2,Y2
    PENNORMAL  :  SET  GTEXTMODE  8
    IF  Flag%=1  THEN  GOTO  Ask_for_letter
    SOUND  :  SOUND
RETURN
```

Mac BASIC

Alphabet

© MacTech   DECEMBER 1984   10



(The following data statements are required by both versions, but are given here for MS basic.)

```
1370 DATA 36,20,52,74          :'A data
1380 DATA 72,20,84,74          :'B data
1390 DATA 103,20,115,74        :'C data
1400 DATA 133,20,147,74        :'D data
1410 DATA 165,20,176,74        :'E data
1420 DATA 195,20,207,74        :'F data
1430 DATA 225,20,239,74        :'G data
1440 DATA 257,20,272,74        :'H data
1450 DATA 290,20,315,74        :'I data
1460 DATA 330,20,350,74        :'J data
1470 DATA 369,20,383,74        :'K data
1480 DATA 401,20,414,74        :'L data
1490 DATA 434,20,454,74        :'M data
1500 DATA 35,75,51,120         :'N data
1510 DATA 69,75,83,120         :'O data
1520 DATA 101,75,115,120       :'P data
1530 DATA 133,75,147,120       :'Q data
1540 DATA 165,75,179,120       :'R data
1550 DATA 197,75,208,120       :'S data
1560 DATA 227,75,240,120       :'T data
1570 DATA 259,75,275,120       :'U data
1580 DATA 294,75,310,120       :'V data
1590 DATA 329,75,352,120       :'W data
1600 DATA 371,75,385,120       :'X data
1610 DATA 402,75,417,120       :'Y data
1620 DATA 434,75,445,120       :'Z data
```

PASCAL PROCEDURES

by
Chris Derossi

"MacPascal Arrives!"

Well, after waiting and waiting and waiting, Macintosh Pascal has finally arrived, and with its arrival, Mac users now have a serious high-level language for programming on the Macintosh. This column will help the users of Macintosh Pascal both to learn about Mac Pascal, and to learn how the wonderous abilities of the Mac ROM can tie into Mac Pascal for creating powerful and useful programs.

For this opening column, we'll be taking a general look at Pascal, and we'll see how you can get started right away learning how to use your Pascal to the fullest. Let us start out by examining the contents of your Pascal package.

The documentation is in three seperate parts, with each section contained in its own little booklet. The first, and smallest book contains the standard instructions for using the normal Macintosh User Interfaces, to which Mac Pascal conforms. For experienced users of the Mac, this book will be of very little help; you already know all about clicking, dragging, and editing with the mouse. For the less experienced user, this book will teach you what you need to know in order to get started with Pascal right away.

The most important part of this book is the section which describes the use of the Observe and Instant windows. This is the only explanantion of these windows found in the documentation. As we get further into Pascal, the importance of these windows will grow.

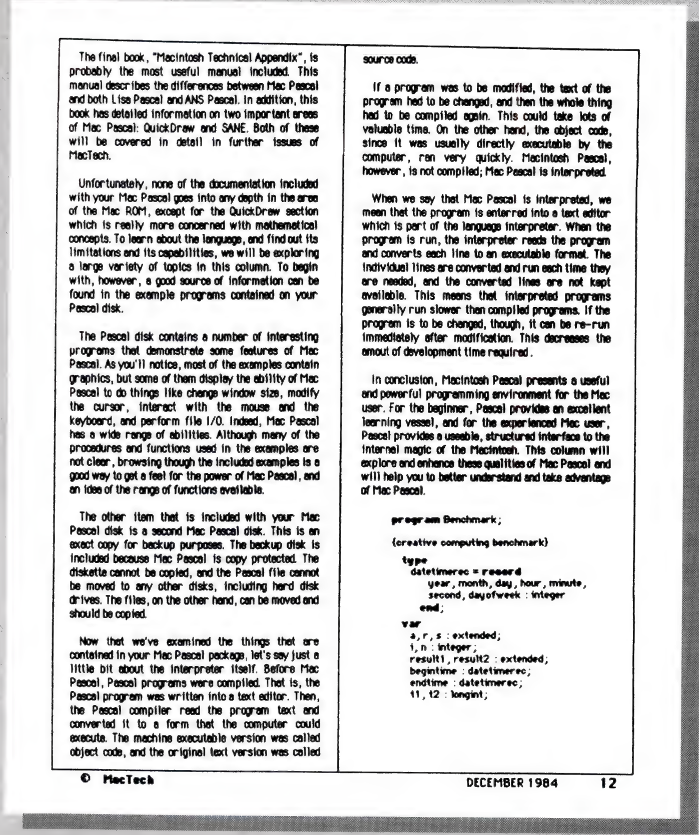
The booklet titled "Pascal Reference Manual" is an important tool for beginners and experienced programmers alike. Although it is not a tutorial, it does contain the entire syntax specification for Mac Pascal. Its primary use is as a look-up reference. New Pascal programmers may wish to go through the manual to get a general feel for the language, but as I'll discuss shortly, more can be learned by studying the example programs supplied with Mac Pascal.

Text

```
time in seconds=        19
 accuracy=0.000000000000005662
 random= 1.16641700000e+6
```

Figure 1 shows the output of the Ahl benchmark published in creative computing. Notice the high degree of accuracy in only 19 seconds. This compares to MS Basic time of 1 minute, 36 seconds on the Mac. The random time does not compare due to the difference in implementation of the random number generator.

© MacTech   DECEMBER 1984   11



The final book, "Macintosh Technical Appendix", is probably the most useful manual included. This manual describes the differences between Mac Pascal and both Lisa Pascal and ANS Pascal. In addition, this book has detailed information on two important areas of Mac Pascal: QuickDraw and SANE. Both of these will be covered in detail in further issues of MacTech.

Unfortunately, none of the documentation included with your Mac Pascal goes into any depth in the area of the Mac ROM, except for the QuickDraw section which is really more concerned with mathematical concepts. To learn about the language, and find out its limitations and its capabilities, we will be exploring a large variety of topics in this column. To begin with, however, a good source of information can be found in the example programs contained on your Pascal disk.

The Pascal disk contains a number of interesting programs that demonstrate some features of Mac Pascal. As you'll notice, most of the examples contain graphics, but some of them display the ability of Mac Pascal to do things like change window size, modify the cursor, interact with the mouse and the keyboard, and perform file I/O. Indeed, Mac Pascal has a wide range of abilities. Although many of the procedures and functions used in the examples are not clear, browsing though the included examples is a good way to get a feel for the power of Mac Pascal, and an idea of the range of functions available.

The other item that is included with your Mac Pascal disk is a second Mac Pascal disk. This is an exact copy for backup purposes. The backup disk is included because Mac Pascal is copy protected. The diskette cannot be copied, and the Pascal file cannot be moved to any other disks, including hard disk drives. The files, on the other hand, can be moved and should be copied.

Now that we've examined the things that are contained in your Mac Pascal package, let's say just a little bit about the interpreter itself. Before Mac Pascal, Pascal programs were compiled. That is, the Pascal program was written into a text editor. Then, the Pascal compiler read the program text and converted it to a form that the computer could execute. The machine executable version was called object code, and the original text version was called source code.

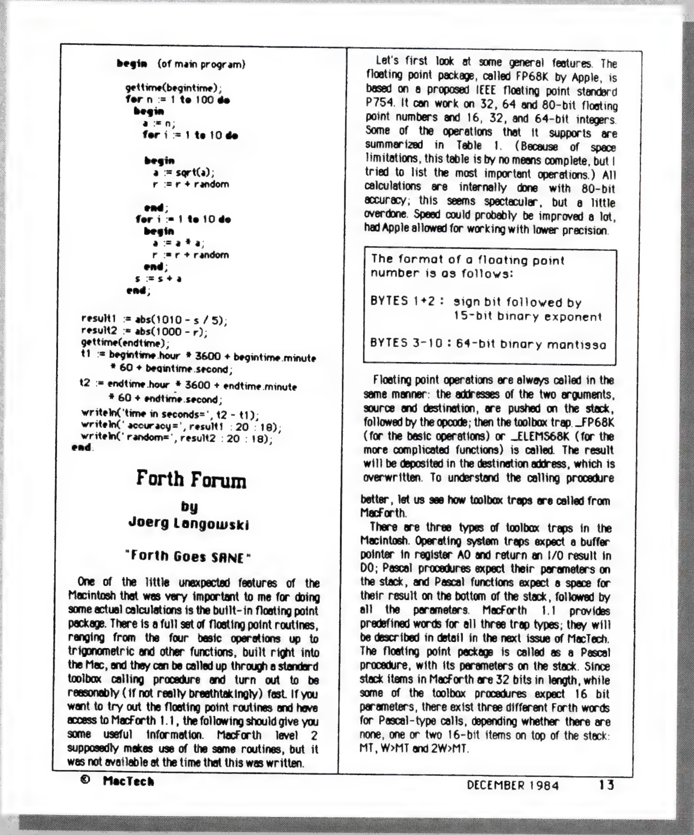
If a program was to be modified, the text of the program had to be changed, and then the whole thing had to be compiled again. This could take lots of valuable time. On the other hand, the object code, since it was usually directly executable by the computer, ran very quickly. Macintosh Pascal, however, is not compiled; Mac Pascal is interpreted.

When we say that Mac Pascal is interpreted, we mean that the program is enterred into a text editor which is part of the language interpreter. When the program is run, the interpreter reads the program and converts each line to an executable format. The individual lines are converted and run each time they are needed, and the converted lines are not kept available. This means that interpreted programs generally run slower than compiled programs. If the program is to be changed, though, it can be re-run immediately after modification. This decreases the amout of development time required.

In conclusion, Macintosh Pascal presents a useful and powerful programming environment for the Mac user. For the beginner, Pascal provides an excellent learning vessel, and for the experienced Mac user, Pascal provides a useable, structured interface to the internal magic of the Macintosh. This column will explore and enhance these qualities of Mac Pascal and will help you to better understand and take advantage of Mac Pascal.

```
program Benchmark;

{creative computing benchmark}

  type
    datetimerec = record
        year, month, day, hour, minute,
        second, dayofweek : integer
      end;

  var
    a, r, s : extended;
    i, n : integer;
    result1, result2 : extended;
    begintime : datetimerec;
    endtime : datetimerec;
    t1, t2 : longint;
```

© MacTech   DECEMBER 1984   12



```
        begin  (of main program)

          gettime(begintime);
          for n := 1 to 100 do
            begin
              a := n;
              for i := 1 to 10 do

              begin
                a := sqrt(a);
                r := r + random

              end;
            for i := 1 to 10 do
              begin
                a := a * a;
                r := r + random
              end;
            s := s + a
          end;

  result1 := abs(1010 - s / 5);
  result2 := abs(1000 - r);
  gettime(endtime);
  t1 := begintime.hour * 3600 + begintime.minute
        * 60 + begintime.second;
  t2 := endtime.hour * 3600 + endtime.minute
        * 60 + endtime.second;
  writeln('time in seconds=', t2 - t1);
  writeln('accuracy=', result1 : 20 : 18);
  writeln('random=', result2 : 20 : 18);
end.
```

Forth Forum

by
Joerg Langowski

"Forth Goes SANE"

One of the little unexpected features of the Macintosh that was very important to me for doing some actual calculations is the built-in floating point package. There is a full set of floating point routines, ranging from the four basic operations up to trigonometric and other functions, built right into the Mac, and they can be called up through a standard toolbox calling procedure and turn out to be reasonably (if not really breathtakingly) fast. If you want to try out the floating point routines and have access to MacForth 1.1, the following should give you some useful information. MacForth level 2 supposedly makes use of the same routines, but it was not available at the time that this was written.

Let's first look at some general features. The floating point package, called FP68K by Apple, is based on a proposed IEEE floating point standard P754. It can work on 32, 64 and 80-bit floating point numbers and 16, 32, and 64-bit integers. Some of the operations that it supports are summarized in Table 1. (Because of space limitations, this table is by no means complete, but I tried to list the most important operations.) All calculations are internally done with 80-bit accuracy; this seems spectacular, but a little overdone. Speed could probably be improved a lot, had Apple allowed for working with lower precision.

The format of a floating point number is as follows:

BYTES 1+2 : sign bit followed by
             15-bit binary exponent

BYTES 3-10 : 64-bit binary mantissa

Floating point operations are always called in the same manner: the addresses of the two arguments, source and destination, are pushed on the stack, followed by the opcode; then the toolbox trap _FP68K (for the basic operations) or _ELEMS68K (for the more complicated functions) is called. The result will be deposited in the destination address, which is overwritten. To understand the calling procedure better, let us see how toolbox traps are called from MacForth.

There are three types of toolbox traps in the Macintosh. Operating system traps expect a buffer pointer in register A0 and return an I/O result in D0; Pascal procedures expect their parameters on the stack, and Pascal functions expect a space for their result on the bottom of the stack, followed by all the parameters. MacForth 1.1 provides predefined words for all three trap types; they will be described in detail in the next issue of MacTech. The floating point package is called as a Pascal procedure, with its parameters on the stack. Since stack items in MacForth are 32 bits in length, while some of the toolbox procedures expect 16 bit parameters, there exist three different Forth words for Pascal-type calls, depending whether there are none, one or two 16-bit items on top of the stack: MT, W>MT and 2W>MT.

© MacTech   DECEMBER 1984   13

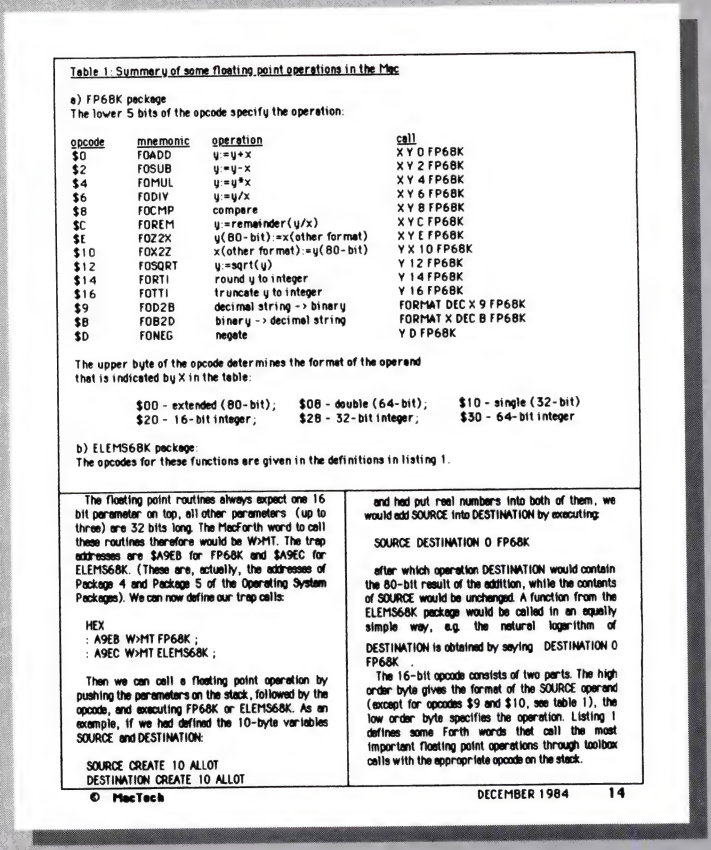


Table 1: Summary of some floating point operations in the Mac

a) FP68K package
The lower 5 bits of the opcode specify the operation:

opcode	mnemonic	operation	call
$0	FOADD	y:=y+x	X Y 0 FP68K
$2	FOSUB	y:=y-x	X Y 2 FP68K
$4	FOMUL	y:=y*x	X Y 4 FP68K
$6	FODIV	y:=y/x	X Y 6 FP68K
$8	FOCMP	compare	X Y 8 FP68K
$C	FOREM	y:=remainder(y/x)	X Y C FP68K
$E	FOZ2X	y(80-bit):=x(other format)	X Y E FP68K
$10	FOX2Z	x(other format):=y(80-bit)	Y X 10 FP68K
$12	FOSQRT	y:=sqrt(y)	Y 12 FP68K
$14	FORTI	round y to integer	Y 14 FP68K
$16	FOTTI	truncate y to integer	Y 16 FP68K
$9	FOD2B	decimal string -> binary	FORMAT DEC X 9 FP68K
$B	FOB2D	binary -> decimal string	FORMAT X DEC B FP68K
$D	FONEG	negate	Y D FP68K

The upper byte of the opcode determines the format of the operand that is indicated by X in the table:

```
$00 - extended (80-bit);   $08 - double (64-bit);   $10 - single (32-bit)
$20 - 16-bit integer;      $28 - 32-bit integer;    $30 - 64-bit integer
```

b) ELEMS68K package:
The opcodes for these functions are given in the definitions in listing 1.

The floating point routines always expect one 16 bit parameter on top, all other parameters (up to three) are 32 bits long. The MacForth word to call these routines therefore would be W>MT. The trap addresses are $A9EB for FP68K and $A9EC for ELEMS68K. (These are, actually, the addresses of Package 4 and Package 5 of the Operating System Packages). We can now define our trap calls:

```
HEX
: A9EB W>MT FP68K ;
: A9EC W>MT ELEMS68K ;
```

Then we can call a floating point operation by pushing the parameters on the stack, followed by the opcode, and executing FP68K or ELEMS68K. As an example, if we had defined the 10-byte variables SOURCE and DESTINATION:

```
SOURCE CREATE 10 ALLOT
DESTINATION CREATE 10 ALLOT
```

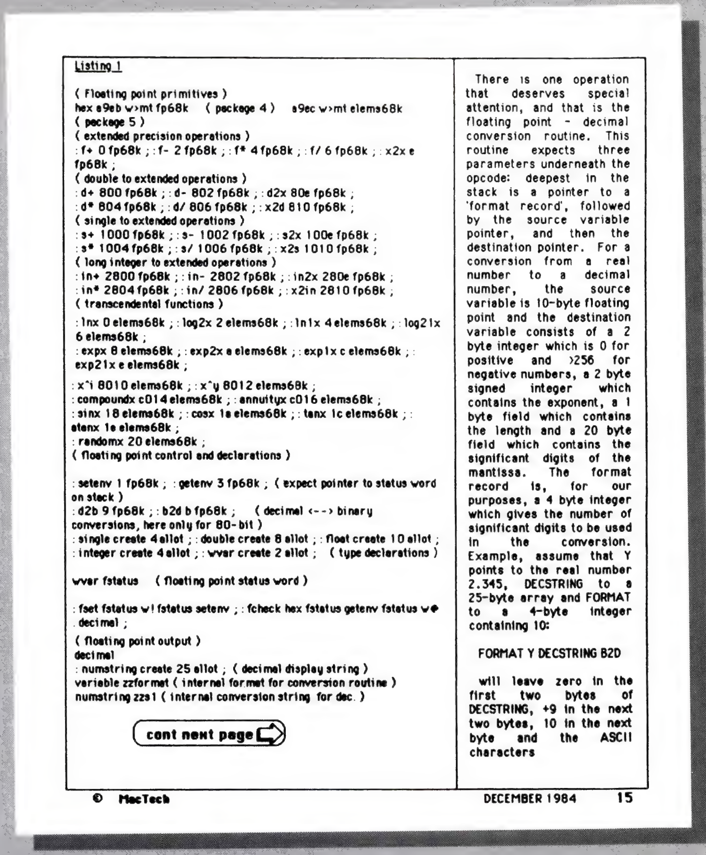
and had put real numbers into both of them, we would add SOURCE into DESTINATION by executing:

```
SOURCE DESTINATION 0 FP68K
```

after which operation DESTINATION would contain the 80-bit result of the addition, while the contents of SOURCE would be unchanged. A function from the ELEMS68K package would be called in an equally simple way, e.g. the natural logarithm of DESTINATION is obtained by saying DESTINATION 0 FP68K .

The 16-bit opcode consists of two parts. The high order byte gives the format of the SOURCE operand (except for opcodes $9 and $10, see table 1), the low order byte specifies the operation. Listing 1 defines some Forth words that call the most important floating point operations through toolbox calls with the appropriate opcode on the stack.

© MacTech DECEMBER 1984 14



Listing 1

```
( Floating point primitives )
hex a9eb w>mt fp68k   ( package 4 )   a9ec w>mt elems68k
( package 5 )
( extended precision operations )
: f+ 0 fp68k ; : f- 2 fp68k ; : f* 4 fp68k ; : f/ 6 fp68k ; : x2x e
fp68k ;
( double to extended operations )
: d+ 800 fp68k ; : d- 802 fp68k ; : d2x 80e fp68k ;
: d* 804 fp68k ; : d/ 806 fp68k ; : x2d 810 fp68k ;
( single to extended operations )
: s+ 1000 fp68k ; : s- 1002 fp68k ; : s2x 100e fp68k ;
: s* 1004 fp68k ; : s/ 1006 fp68k ; : x2s 1010 fp68k ;
( long integer to extended operations )
: in+ 2800 fp68k ; : in- 2802 fp68k ; : in2x 280e fp68k ;
: in* 2804 fp68k ; : in/ 2806 fp68k ; : x2in 2810 fp68k ;
( transcendental functions )

: lnx 0 elems68k ; : log2x 2 elems68k ; : ln1x 4 elems68k ; : log21x
6 elems68k ;
: expx 8 elems68k ; : exp2x a elems68k ; : exp1x c elems68k ; :
exp21x e elems68k ;

: x^i 8010 elems68k ; : x^y 8012 elems68k ;
: compoundx c014 elems68k ; : annuityx c016 elems68k ;
: sinx 18 elems68k ; : cosx 1a elems68k ; : tanx 1c elems68k ; :
atanx 1e elems68k ;
: randomx 20 elems68k ;
( floating point control and declarations )

: setenv 1 fp68k ; : getenv 3 fp68k ; ( expect pointer to status word
on stack )
: d2b 9 fp68k ; : b2d b fp68k ;    ( decimal <--> binary
conversions, here only for 80-bit )
: single create 4 allot ; : double create 8 allot ; : float create 10 allot ;
: integer create 4 allot ; : wvar create 2 allot ;   ( type declarations )

wvar fstatus   ( floating point status word )

: fset fstatus w! fstatus setenv ; : fcheck hex fstatus getenv fstatus w@
. decimal ;
( floating point output )
decimal
: numstring create 25 allot ; ( decimal display string )
variable zzformat ( internal format for conversion routine )
numstring zzs1 ( internal conversion string for dec. )
```

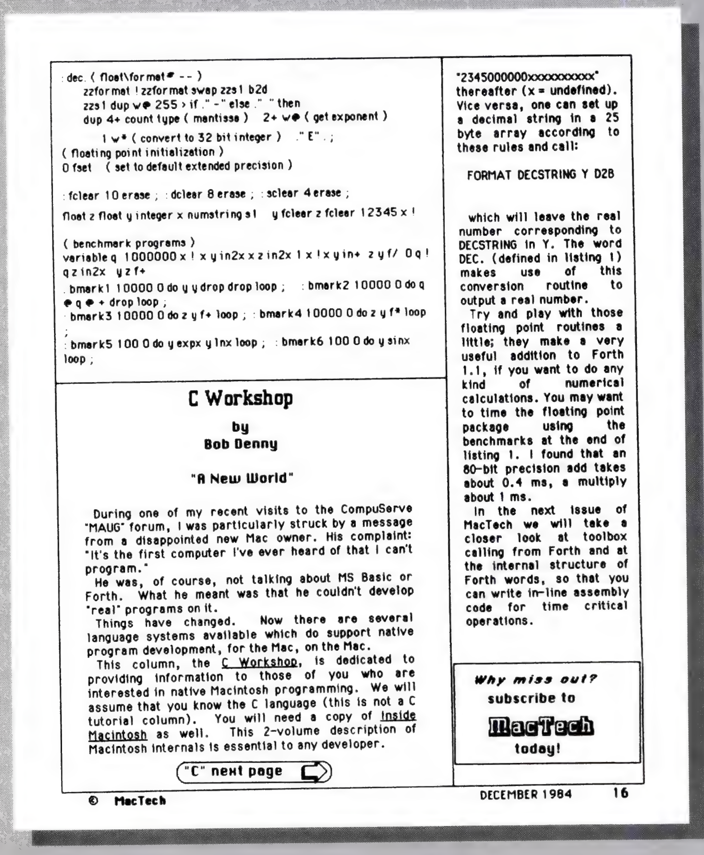
cont next page

There is one operation that deserves special attention, and that is the floating point - decimal conversion routine. This routine expects three parameters underneath the opcode: deepest in the stack is a pointer to a 'format record', followed by the source variable pointer, and then the destination pointer. For a conversion from a real number to a decimal number, the source variable is 10-byte floating point and the destination variable consists of a 2 byte integer which is 0 for positive and >256 for negative numbers, a 2 byte signed integer which contains the exponent, a 1 byte field which contains the length and a 20 byte field which contains the significant digits of the mantissa. The format record is, for our purposes, a 4 byte integer which gives the number of significant digits to be used in the conversion. Example, assume that Y points to the real number 2.345, DECSTRING to a 25-byte array and FORMAT to a 4-byte integer containing 10:

```
FORMAT Y DECSTRING B2D
```

will leave zero in the first two bytes of DECSTRING, +9 in the next two bytes, 10 in the next byte and the ASCII characters

© MacTech DECEMBER 1984 15



```
: dec. ( float\format# -- )
    zzformat ! zzformat swap zzs1 b2d
    zzs1 dup w@ 255 > if ." -" else ."  " then
    dup 4+ count type ( mantissa )   2+ w@ ( get exponent )
      1 w* ( convert to 32 bit integer )   ." E" . ;
( floating point initialization )
0 fset   ( set to default extended precision )

: fclear 10 erase ; : dclear 8 erase ; : sclear 4 erase ;

float z float y integer x numstring s1   y fclear z fclear 12345 x !

( benchmark programs )
variable q  1000000 x ! x y in2x x z in2x 1 x ! x y in+  z y f/  0 q !
q z in2x   y z f+
: bmark1 10000 0 do y y drop drop loop ;   : bmark2 10000 0 do q
@ q @ + drop loop ;
: bmark3 10000 0 do z y f+ loop ; : bmark4 10000 0 do z y f* loop
;
: bmark5 100 0 do y expx y lnx loop ;  : bmark6 100 0 do y sinx
loop ;
```

"2345000000xxxxxxxxxx" thereafter (x = undefined). Vice versa, one can set up a decimal string in a 25 byte array according to these rules and call:

```
FORMAT DECSTRING Y D2B
```

which will leave the real number corresponding to DECSTRING in Y. The word DEC. (defined in listing 1) makes use of this conversion routine to output a real number.

Try and play with those floating point routines a little; they make a very useful addition to Forth 1.1, if you want to do any kind of numerical calculations. You may want to time the floating point package using the benchmarks at the end of listing 1. I found that an 80-bit precision add takes about 0.4 ms, a multiply about 1 ms.

In the next issue of MacTech we will take a closer look at toolbox calling from Forth and at the internal structure of Forth words, so that you can write in-line assembly code for time critical operations.

C Workshop

by
Bob Denny

"A New World"

During one of my recent visits to the CompuServe "MAUG" forum, I was particularly struck by a message from a disappointed new Mac owner. His complaint: "It's the first computer I've ever heard of that I can't program."

He was, of course, not talking about MS Basic or Forth. What he meant was that he couldn't develop "real" programs on it.

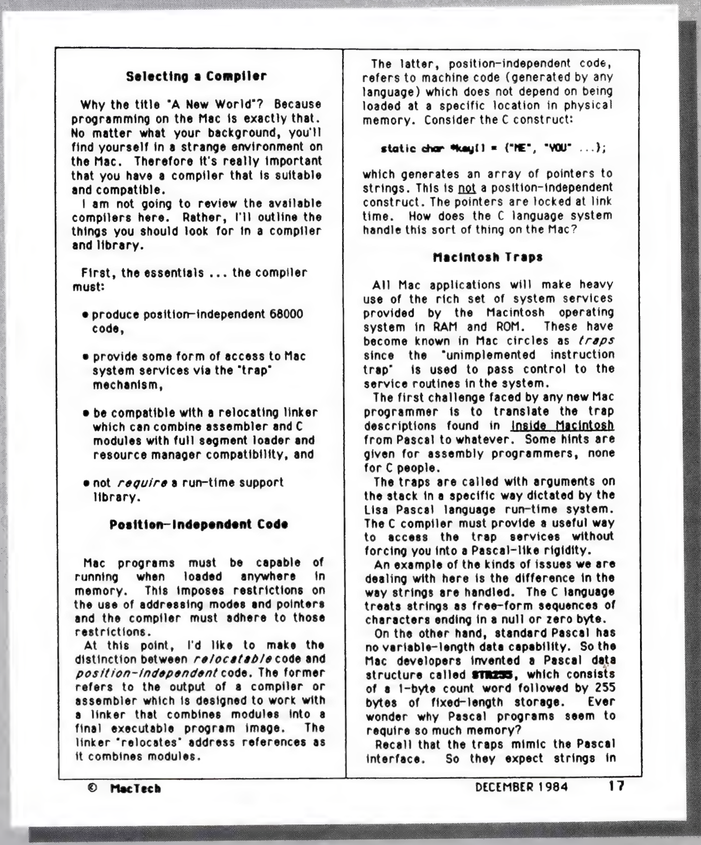
Things have changed. Now there are several language systems available which do support native program development, for the Mac, on the Mac.

This column, the C Workshop, is dedicated to providing information to those of you who are interested in native Macintosh programming. We will assume that you know the C language (this is not a C tutorial column). You will need a copy of Inside Macintosh as well. This 2-volume description of Macintosh internals is essential to any developer.

"C" next page

Why miss out?
subscribe to
MacTech
today!

© MacTech DECEMBER 1984 16



Selecting a Compiler

Why the title "A New World"? Because programming on the Mac is exactly that. No matter what your background, you'll find yourself in a strange environment on the Mac. Therefore it's really important that you have a compiler that is suitable and compatible.

I am not going to review the available compilers here. Rather, I'll outline the things you should look for in a compiler and library.

First, the essentials ... the compiler must:

- produce position-independent 68000 code,
- provide some form of access to Mac system services via the "trap" mechanism,
- be compatible with a relocating linker which can combine assembler and C modules with full segment loader and resource manager compatibility, and
- not *require* a run-time support library.

Position-Independent Code

Mac programs must be capable of running when loaded anywhere in memory. This imposes restrictions on the use of addressing modes and pointers and the compiler must adhere to those restrictions.

At this point, I'd like to make the distinction between *relocatable* code and *position-independent* code. The former refers to the output of a compiler or assembler which is designed to work with a linker that combines modules into a final executable program image. The linker "relocates" address references as it combines modules.

The latter, position-independent code, refers to machine code (generated by any language) which does not depend on being loaded at a specific location in physical memory. Consider the C construct:

```
static char *key[] = {"ME", "YOU" ...};
```

which generates an array of pointers to strings. This is not a position-independent construct. The pointers are locked at link time. How does the C language system handle this sort of thing on the Mac?

Macintosh Traps

All Mac applications will make heavy use of the rich set of system services provided by the Macintosh operating system in RAM and ROM. These have become known in Mac circles as *traps* since the "unimplemented instruction trap" is used to pass control to the service routines in the system.

The first challenge faced by any new Mac programmer is to translate the trap descriptions found in Inside Macintosh from Pascal to whatever. Some hints are given for assembly programmers, none for C people.

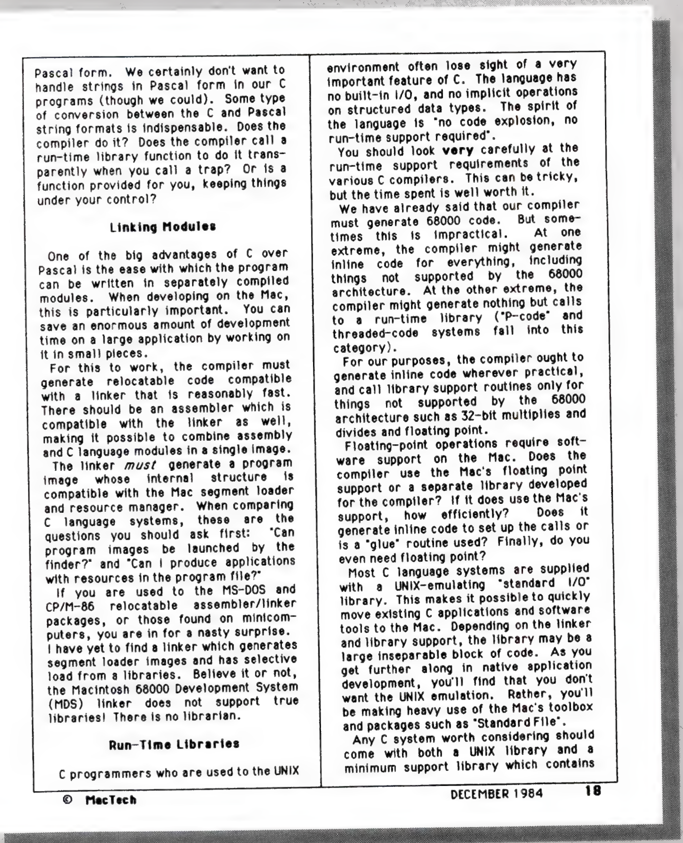
The traps are called with arguments on the stack in a specific way dictated by the Lisa Pascal language run-time system. The C compiler must provide a useful way to access the trap services without forcing you into a Pascal-like rigidity.

An example of the kinds of issues we are dealing with here is the difference in the way strings are handled. The C language treats strings as free-form sequences of characters ending in a null or zero byte.

On the other hand, standard Pascal has no variable-length data capability. So the Mac developers invented a Pascal data structure called **STR255**, which consists of a 1-byte count word followed by 255 bytes of fixed-length storage. Ever wonder why Pascal programs seem to require so much memory?

Recall that the traps mimic the Pascal interface. So they expect strings in

© MacTech DECEMBER 1984 17



Pascal form. We certainly don't want to handle strings in Pascal form in our C programs (though we could). Some type of conversion between the C and Pascal string formats is indispensable. Does the compiler do it? Does the compiler call a run-time library function to do it transparently when you call a trap? Or is a function provided for you, keeping things under your control?

Linking Modules

One of the big advantages of C over Pascal is the ease with which the program can be written in separately compiled modules. When developing on the Mac, this is particularly important. You can save an enormous amount of development time on a large application by working on it in small pieces.

For this to work, the compiler must generate relocatable code compatible with a linker that is reasonably fast. There should be an assembler which is compatible with the linker as well, making it possible to combine assembly and C language modules in a single image.

The linker *must* generate a program image whose internal structure is compatible with the Mac segment loader and resource manager. When comparing C language systems, these are the questions you should ask first: "Can program images be launched by the finder?" and "Can I produce applications with resources in the program file?"

If you are used to the MS-DOS and CP/M-86 relocatable assembler/linker packages, or those found on minicomputers, you are in for a nasty surprise. I have yet to find a linker which generates segment loader images and has selective load from a libraries. Believe it or not, the Macintosh 68000 Development System (MDS) linker does not support true libraries! There is no librarian.

Run-Time Libraries

C programmers who are used to the UNIX environment often lose sight of a very important feature of C. The language has no built-in I/O, and no implicit operations on structured data types. The spirit of the language is "no code explosion, no run-time support required".

You should look **very** carefully at the run-time support requirements of the various C compilers. This can be tricky, but the time spent is well worth it.

We have already said that our compiler must generate 68000 code. But sometimes this is impractical. At one extreme, the compiler might generate inline code for everything, including things not supported by the 68000 architecture. At the other extreme, the compiler might generate nothing but calls to a run-time library ("P-code" and threaded-code systems fall into this category).

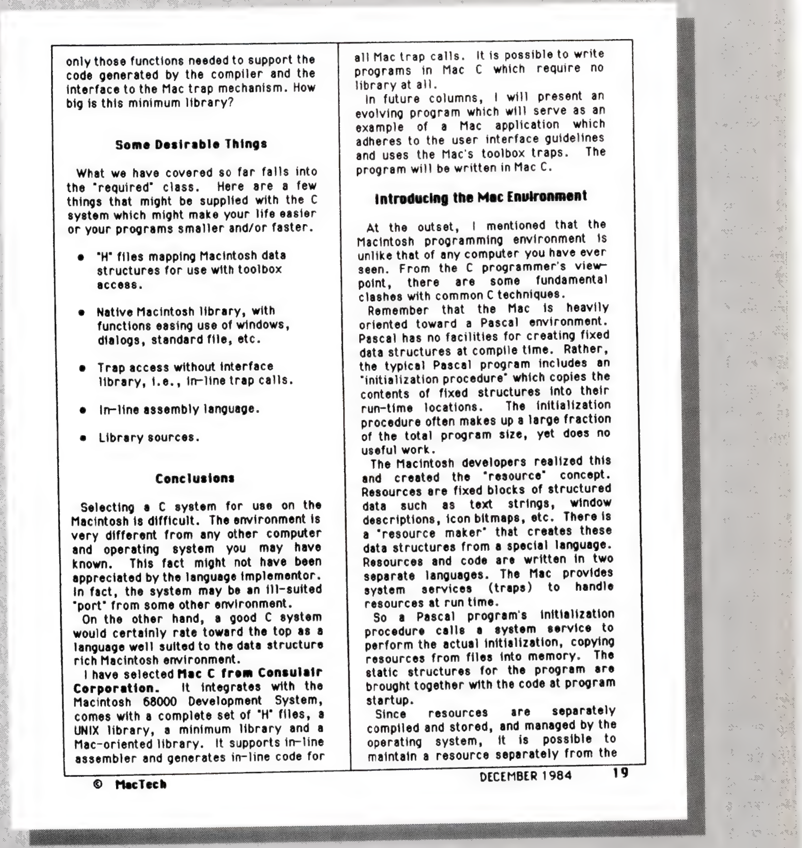
For our purposes, the compiler ought to generate inline code wherever practical, and call library support routines only for things not supported by the 68000 architecture such as 32-bit multiplies and divides and floating point.

Floating-point operations require software support on the Mac. Does the compiler use the Mac's floating point support or a separate library developed for the compiler? If it does use the Mac's support, how efficiently? Does it generate inline code to set up the calls or is a "glue" routine used? Finally, do you even need floating point?

Most C language systems are supplied with a UNIX-emulating "standard I/O" library. This makes it possible to quickly move existing C applications and software tools to the Mac. Depending on the linker and library support, the library may be a large inseparable block of code. As you get further along in native application development, you'll find that you don't want the UNIX emulation. Rather, you'll be making heavy use of the Mac's toolbox and packages such as "Standard File".

Any C system worth considering should come with both a UNIX library and a minimum support library which contains

© **MacTech** DECEMBER 1984 **18**



only those functions needed to support the code generated by the compiler and the interface to the Mac trap mechanism. How big is this minimum library?

Some Desirable Things

What we have covered so far falls into the "required" class. Here are a few things that might be supplied with the C system which might make your life easier or your programs smaller and/or faster.

- "H" files mapping Macintosh data structures for use with toolbox access.
- Native Macintosh library, with functions easing use of windows, dialogs, standard file, etc.
- Trap access without interface library, i.e., in-line trap calls.
- In-line assembly language.
- Library sources.

Conclusions

Selecting a C system for use on the Macintosh is difficult. The environment is very different from any other computer and operating system you may have known. This fact might not have been appreciated by the language implementor. In fact, the system may be an ill-suited "port" from some other environment.

On the other hand, a good C system would certainly rate toward the top as a language well suited to the data structure rich Macintosh environment.

I have selected **Mac C from Consulair Corporation.** It integrates with the Macintosh 68000 Development System, comes with a complete set of "H" files, a UNIX library, a minimum library and a Mac-oriented library. It supports in-line assembler and generates in-line code for all Mac trap calls. It is possible to write programs in Mac C which require no library at all.

In future columns, I will present an evolving program which will serve as an example of a Mac application which adheres to the user interface guidelines and uses the Mac's toolbox traps. The program will be written in Mac C.

Introducing the Mac Environment

At the outset, I mentioned that the Macintosh programming environment is unlike that of any computer you have ever seen. From the C programmer's viewpoint, there are some fundamental clashes with common C techniques.

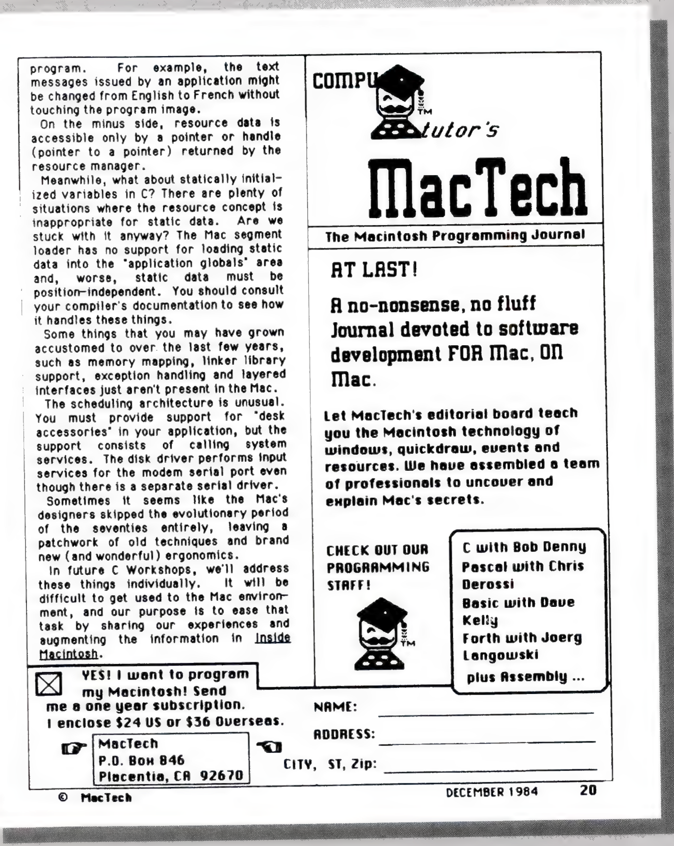
Remember that the Mac is heavily oriented toward a Pascal environment. Pascal has no facilities for creating fixed data structures at compile time. Rather, the typical Pascal program includes an "initialization procedure" which copies the contents of fixed structures into their run-time locations. The initialization procedure often makes up a large fraction of the total program size, yet does no useful work.

The Macintosh developers realized this and created the "resource" concept. Resources are fixed blocks of structured data such as text strings, window descriptions, icon bitmaps, etc. There is a "resource maker" that creates these data structures from a special language. Resources and code are written in two separate languages. The Mac provides system services (traps) to handle resources at run time.

So a Pascal program's initialization procedure calls a system service to perform the actual initialization, copying resources from files into memory. The static structures for the program are brought together with the code at program startup.

Since resources are separately compiled and stored, and managed by the operating system, it is possible to maintain a resource separately from the

© **MacTech** DECEMBER 1984 **19**



program. For example, the text messages issued by an application might be changed from English to French without touching the program image.

On the minus side, resource data is accessible only by a pointer or handle (pointer to a pointer) returned by the resource manager.

Meanwhile, what about statically initialized variables in C? There are plenty of situations where the resource concept is inappropriate for static data. Are we stuck with it anyway? The Mac segment loader has no support for loading static data into the "application globals" area and, worse, static data must be position-independent. You should consult your compiler's documentation to see how it handles these things.

Some things that you may have grown accustomed to over the last few years, such as memory mapping, linker library support, exception handling and layered interfaces just aren't present in the Mac.

The scheduling architecture is unusual. You must provide support for "desk accessories" in your application, but the support consists of calling system services. The disk driver performs input services for the modem serial port even though there is a separate serial driver.

Sometimes it seems like the Mac's designers skipped the evolutionary period of the seventies entirely, leaving a patchwork of old techniques and brand new (and wonderful) ergonomics.

In future C Workshops, we'll address these things individually. It will be difficult to get used to the Mac environment, and our purpose is to ease that task by sharing our experiences and augmenting the information in Inside Macintosh.

COMPUtutor's™

MacTech

The Macintosh Programming Journal

AT LAST!

A no-nonsense, no fluff Journal devoted to software development FOR Mac, ON Mac.

Let MacTech's editorial board teach you the Macintosh technology of windows, quickdraw, events and resources. We have assembled a team of professionals to uncover and explain Mac's secrets.

CHECK OUT OUR PROGRAMMING STAFF!

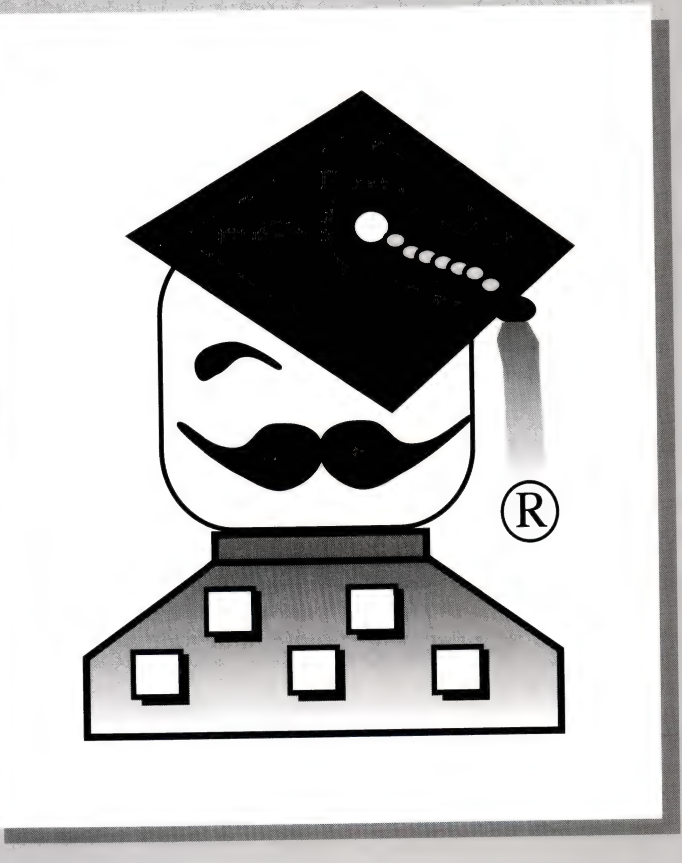
C with Bob Denny
Pascal with Chris Derossi
Basic with Dave Kelly
Forth with Joerg Langowski
plus Assembly ...

☒ **YES! I want to program my Macintosh! Send me a one year subscription. I enclose $24 US or $36 Overseas.**

MacTech
P.O. Box 846
Placentia, CA 92670

NAME:	
ADDRESS:	
CITY, ST, Zip:	

© **MacTech** DECEMBER 1984 **20**



The Editor's Page continued from page 4

share on the Macintosh as they could, they see expanding into the Windows market as a sensible way to grow the company. Besides, by the time they've grown to this point, they have customers clamoring for a Windows version. In one case, it's become more than twice as hard to maintain a product on two platforms (what do you do when a Mac technology isn't available on Windows?), with no more revenue than expected staying with Macintosh alone. They owned their market; people bought Macs just to use their product. Putting it on Windows gave those people a choice of platform, but didn't draw in new customers. Not all products enjoy this position, so mileage no doubt varies from company to company.

We'd like to hear from you. Let's get some of your stories about comings, goings, reasons, and outcomes.

Sorry About The Confusion

Last month I wrote about EvenBetterBusError – what it is, what it does, and why software that doesn't work with it is buggy. Unfortunately, we experienced a glitch in the prepress process, and wound up with source code lines that wrapped, making the assembly source listing confusing and hard to read. Worse yet, two good quotes got pushed off the page!

Since the issue of reading from or writing to memory location 0 is so important, here it is again. Greg Marriott wrote EvenBetterBusError (aka EBBE). It puts a 4-byte value into memory location 0 which will generate a bus error or an illegal instruction on any Macintosh if someone dereferences a nil pointer or if they jump to 0. EBBE periodically checks to see whether someone has written to location 0 (probably using a NIL pointer). If so, it drops into the debugger and says, "Write to NIL." EBBE can help you make your software better.

Here's reconstituted code which shows you everything that EvenBetterBusError does. It's pretty simple. It installs a VBL task which periodically checks location 0, drops into the debugger if it needs to, and stuffs $50FF8001 back into location 0.

It's *never* ok to write to location 0, nor is it ever ok to dereference a NIL pointer. Code that does these things is *buggy*.

```
        BRA.S    InstallVBL
BeginCodeBlock
VBLRecord
        DC.L 00000000,0001    ; QElemPtr,  queue type (1==VBL queue)
        DC.L 00000000         ; ptr to VBL code
        DC.W 0001,0000        ; timeout count & phase count
        MOVE.L   $0,D0                  ; put location 0's contents into D0
        ANDI.L   #$7FFFFFFF,D0          ; strip the high bit
        CMPI.L   #$50FF8001,D0          ; see if someone wrote over it
        BEQ.S    SkipDebugStr           ; if not, everything's ok
        CMPI.L   #$40810000,D0          ; see if it's ProcMgr's safe value
        BEQ.S    SkipDebugStr           ; if so, everything's ok
        PEA      BuggyCodeWroteToNIL
        _DebugStr
SkipDebugStr
        MOVE.L   #$50FF8001,$0          ; put a bus error value at $0
        MOVE.W   #$0001,$000A(A0)       ; reset the VBL timer
        RTS
EndOfCodeBlock

BuggyCodeWroteToNIL   DC.B 'Write to NIL',00
SizeOfCodeBlock equ  EndOfCodeBlock-BeginCodeBlock

InstallVBL
        MOVEQ    #SizeOfCodeBlock,D0 ; allocate a block for code & data
        _NewPtr   ,sys               ; make a block in the system heap
        MOVE.L   A0,-(A7)            ; remember this block's address
        MOVEA.L  A0,A1
        LEA      VBLRecord,A0        ; a static copy of our VBL record
        MOVEQ    #SizeOfCodeBlock,D0 ; the size of the VBL code & data
        _BlockMove                   ; copy from INIT into sys heap block
        MOVEA.L  (A7)+,A0            ; the system heap block
        PEA      $000E(A0)           ; addr of ptr to VBL in VBL record
        MOVE.L   (A7)+,$0006(A0)     ; stuff ptr to VBL code
        _VInstall
        MOVE.L   #$50FF8001,$0       ; put a bus error value at $0
        RTS
```

Food For Thought

Post mortem debugging can be fun, especially when you get to see the victim die over and over again in slow motion!

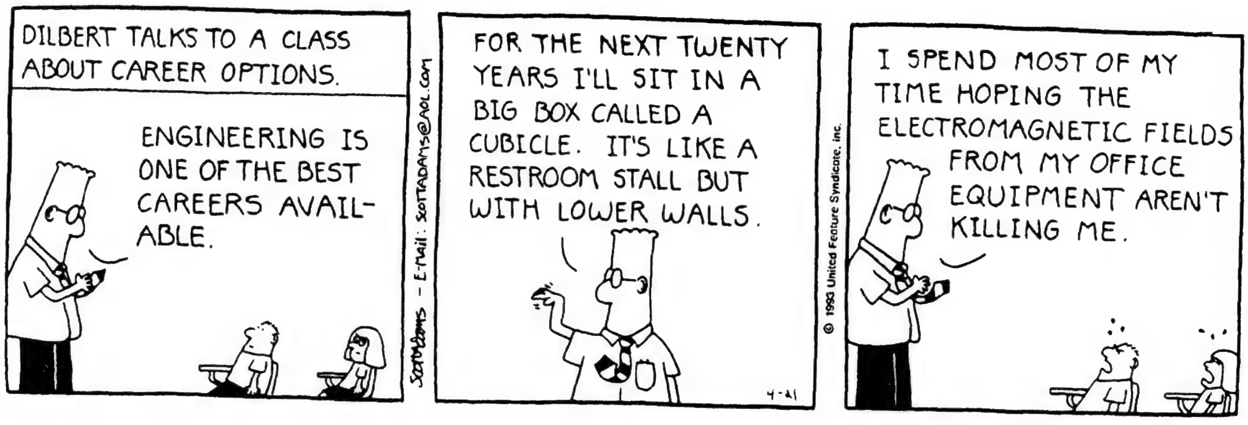
– *Greg Marriott*

Seen On The Net

Macintosh, for those who can see through Windows.

– *reported by baileyc@beetle.com*

Dilbert *by Scott Adams*


DILBERT TALKS TO A CLASS ABOUT CAREER OPTIONS.
ENGINEERING IS ONE OF THE BEST CAREERS AVAILABLE.
ScottAdams - E-Mail: SCOTTADAMS@AOL.COM
FOR THE NEXT TWENTY YEARS I'LL SIT IN A BIG BOX CALLED A CUBICLE. IT'S LIKE A RESTROOM STALL BUT WITH LOWER WALLS.
4-21
© 1993 United Feature Syndicate, Inc.
I SPEND MOST OF MY TIME HOPING THE ELECTROMAGNETIC FIELDS FROM MY OFFICE EQUIPMENT AREN'T KILLING ME.

reprinted by permission of UFS, Inc.

UNIVERSAL RESOURCE LOCATORS

By Scott T Boyd, Editor

Many of you have asked for pointers to interesting places. We've whipped up a quick hodge-podge of points of interest. It's certainly incomplete. If there's something you'd like to see here, please drop us a note at editorial@xplain.com.

In case you're not familiar with Universal Resource Locator (URL) format, it's essentially

```
    <servicekind>://<servername>/<pathname>.
```

That's a bit of an oversimplification (e.g. the path can be far more interesting), but it should be enough to you started.

Products mentioned in this issue

Object Master Universal	(800) 384-0010 or (408) 252-4444
QC	onyxtech@aol.com, demo on AOL

Interesting Developer Places

Dylan	ftp://cambridge.apple.com/pub/dylan
see also	http://legend.gwydion.cs.cmu.edu:8001/dylan
Lisp	http://www.cs.rochester.edu/u/miller/alu.html
Smalltalk	http://www.qks.com
MacTech	ftp://ftp.netcom.com/pub/xplain
OpenDoc	ftp://cil.org
alt.sources.mac	ftp://ftpbio.bgsu.edu/alt.sources.mac
Best of the Net	http://nearnet.gnn.com/gnn/gnn.html
MacGL	ftp://ftp.netcom.com/pub/loceff
Applescript	ftp://gaea.kgs.ukans.edu/applescript
Alpha	ftp://cs.rice.edu:public/Alpha
BBEdit	ftp://ftp.netcom.com:/pub/bbsw
MacNosy	ftp://ftp.netcom.com/pub/macnosy
Symantec	ftp://devtools.symantec.com/Macintosh/Updates/DevTools
TCL stuff	ftp://ics.uci.edu/mac

People & Places

Apple	ftp://ftp.apple.com/
Apple developer	http://www.support.apple.com
Bill Modesitt	ftp://ftp.maui.com/pub/mauisw
Consensus	http://www.consensus.com:8300
info-mac archives	ftp://amug.org/info-mac
	ftp://mac.archive.umich.edu/mac
Paul Robichaux	http://www.iquest.com/~fairgate
Texas A&M	http://gopher.tamu.edu/
Peter Lewis tcp/ip apps	ftp://amug.org/pub/peterlewis

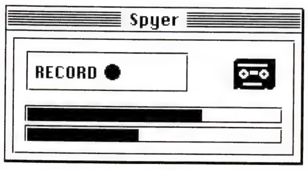
Getting on the net

For those of you with only unix shell account network access, you may want to check out TIA, The Internet Adapter. The Internet Adaptor is a program for unix systems that lets you emulate a SLIP line in software, and that lets you run software like Mosaic and NewsWatcher. Telnet to tia.marketplace.com or point your www browser at marketplace.com for more info.

Spyer

What You Did Is What You See

If you are looking for a simple way to replay your actions - including mouse movement, for software testing or demonstration - Spyer is the answer.



Available: 19236
Recorded: 206
Replayed: 0

Features:
- Records keyboard and mouse actions.
- Stores recorded data in files for future use.
- Controllable Replay speed.
- Works on any Macintosh application.
- A background process triggered by Hot Keys for easy operation.

Price: $ 39 + shipping

In Cider **US:** **Tel:** 718-991-1689 **Fax:** 718-328-8513 **ALink:** INCI.MISHA
International: **Tel:** 972-3-9613530 **Fax:** 972-3-9613827

NOW YOU CAN ADD MOVIE-LIKE
GRAPHIC EFFECTS
TO YOUR MAC APPLICATIONS

QDFx™

•WIPES • DISSOLVES • TRANSITIONS & MORE

Already part of a new multimedia authoring tool, Ariel Publishing's latest release of "QDFx", includes a library of 20 graphic effects you can apply to your Mac applications and without the overhead of including a quick time movie.

It's fast, it's small, it's royalty free and only...

$69.95

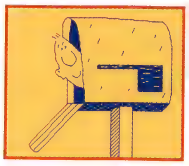
System 7 compatible, 32-bit clean, C source code available (sold separately). Native Power PC version available by June 15, 1994

Ariel Publishing Inc.

To order by mail:	P.O. Box 398 Pateros, WA 98846-0398
Phone:	(509) 923-2249
E-mail:	America Online - (ARIELPUB1), GEnie - (InsideBasic), CIS - (71441,2262)

Visa/MC and purchase orders cheerfully accepted.

By Scott T Boyd, Editor



Sprocket Thoughts

I like the idea of a comparatively tiny object-oriented framework as a base for future articles. But as a beginner in programming (any computer, not just Macintosh) I am a little stuck here. I own neither a PowerMac nor Metrowerks CodeWarrior.

If Sprocket should not be limited to the more advanced users, I (and other hobbyists I guess) would need detailed descriptions how to get Sprocket compiling with THINK C++ 7.0 at least.

Naturally a version for both compilers would still be better, because I would not waste my spare time getting the framework running while the new issue of MacTech Magazine with the next feature article is already in the box.

— *Michael W. Schwarz, Darmstadt (Germany)*
mschwarz@merck.de

We're planning to make Sprocket buildable in all of the major C++ environments. We're also hoping to see other language versions as well. If anyone would like to volunteer, drop us a note at editorial@xplain.com — Ed stb

Prophet of the Apocalypse?

>My co-worker just discovered that you cannot run Excel 5.0
>without OLE installed...In order to display the Windows '95
>(a.k.a. Chicago) logo on your Windows product, you must
>implement OLE 2.0. No OLE, No logo!!!
>What message is MicroSoft sending everyone?

That now that the Justice Department has backed off, it can do whatever it pleases. :-) I recently got a humorous post over the net suggesting that Bill Gates is the Anti-Christ. The best part:

Revelation 13:16 and 13:18 says:

He causes all, both small and great, rich and poor, free and slave, to receive a mark on their right hand or on their foreheads, and that no one may buy or sell except one who has the mark or the name of the beast, or the number of his name.

"Windows compatible?"

— *Bruce F. Webster, CTO, Pages Software*
bwebster@pages.com

Happy To Be Online

Just a note of thanks for having your sources available for Internet ftp. I know it may seem trivial to you, but it's a great service to your readers. I read the mag on a weekend, and I can go get the relevant information *then*... what a win! Keep up the good work.

— *Leo Hourvitz, leo@netcom.com*

Thanks. We're getting things together to get even more useful information online. We'll keep you posted – Ed stb

A Trail of Good Intentions

It was with great interest that I read David Simmons' letter in MacTech Magazine (August 1994) concerning OO languages, Smalltalk, C++, and the design of applications.

Certain sentences leapt from the page:

"I think it is really C++ with its non-dynamic architecture and complex semantics/grammar that is failing".

Absolutely. When I first heard of C++ I looked eagerly for great flexibility in the language and found instead a syntactic nightmare. Jim Gimpel has a quote from Ray Duncan in his manual for C++ FlexeLint, describing C++ as "one of the most grotesque and cryptic languages ever created". It's a mess. It inches asymptotically toward its eventual design while leaving a trail of good intentions.

"Building components in C++ is just as hard as building applications (talk to the folks who are trying to do it). The real problem at hand is managing complexity and capturing design intent."

I keep hawking the "complexity" word to clients when I am trying to get them to write decent code and manage their projects properly. Too many people spend too much time staring at the lines on the screen to see the big picture. Complexity is the one thing that is guaranteed to kill a project as it grows unless it is managed very carefully. Design intent is usually unclear as projects mature and budgets change. Another argument for reducing complexity.

— *Steven Weller, Windsor Systems, Louisville, KY*
steve@barefoot.com
http://iglou.com/~stevenw/windsorhome.html

Reader Report Card

Well, I know you appreciate feedback, so here we go:

I've been a subscriber for almost two years by now. Through that period I've found that MacTech has improved in a lot of ways.

First of all you've done a real good job in covering the mainstream development tools from Symantec and MetroWerks. This is what most of the people I know around here uses, so this is a sound approach in my opinion. Topics like Forth and Fortran are only of academic interest to me (and I guess a lot of other folks).

I'm also very pleased to be able to read detailed information about PPC, Dylan, OpenDoc and other potentially revolutionary stuff lurking just below the horizon. We need to know a little about this and you do a good job of keeping us informed.

Putting the source code from the magazine on your ftp-site is a much appreciated service.

Making more informative folios is also a nice touch. But speaking of layout I would like to suggest that you do something about your usage of color. I think color is a good

Continued on page 86

By Scott T Boyd, Editor

Cross-platform, Royalty-free Electronic Publishing

Foundation Solutions announces royalty-free MultiDoc Power Publisher. It includes fast, comprehensive searching capabilities; efficient handling of large amounts of data without slowing performance; use of existing source material in a variety of formats; and a production process that enables easy revisions to content and interface. A publishing team can build MSWindows and Macintosh multimedia, interactive, electronic publications with customized presentation formats. Microsoft's Encarta 94 Multimedia Encyclopedia and Cinemania were created with MultiDoc Power Publisher.

Evaluation packages are available, as are volume discounts, training courses, and support contracts. Introductory price until February 1, 1995 of $4995 for a single platform package, $6995 for a multi-platform package.

For more info, contact Foundation Solutions at 2000 Regency Parkway, Suite 345, Cary, NC 27511. (800) 247-7890, (919) 481-3517 voice, (919) 481-3551 fax. AppleLink Foundation, CompuServe 71303,1314.

World Wide Web Patent Site

Gregory Aharonian runs a web site with 500 MB of PTO & patenting information, including the beginning of an Internet site that provides full searching capabilities of the Patent & Trademark Office's patent text databases for free. Beyond lots of documents, the current Mosaic site allows you to retreive patent titles in any class/subclass by clicking through a few screens. The service is free. Donations are welcomed. The Mosaic site is at:

http://sunsite.unc.edu/patents/intropat.html

Butt-head Undefined

According to the Media Law Reporter, vol. 22, pp. 2141-2146, the U.S. District Court, Central District of California, has dismissed a libel suit against Apple Computer brought by astronomer Carl Sagan.

In 1993, Apple had a project code-named Carl Sagan. Dr. Sagan made a legal request that they quit doing so. Then, according to the court's opinion, Apple started calling the project "butt-head astronomer."

Judge J. Baird writes: "Plantiff's libel action is based on the allegation that Defendant changed the 'code name' on its personal computer from 'Carl Sagan' to 'Butt-Head Astronomer' after plaintiff had request that Defendant cease use of Plaintiff's name… There can be no question that the use of the figurative term 'Butt-Head' negates the impression that Defendant was seriously implying an assertion of fact. It strains reason to conclude that Defendant was attempting to criticize Plaintiff's reputation or competency as an astronomer. One does not seriously attack the expertise of a scientist using the undefined phrase 'butt-head.' Thus, the figurative language militates against implying an assertion of fact…"

"Furthermore, the tenor of any communication of the information, especially the phrase 'Butt-Head Astronomer,' would negate the impression that Defendant was implying an assertion of fact."

SmalltalkAgents Supports 7.5

Quasar Knowledge Systems, (QKS) Inc. announced that the current version of SmalltalkAgents supports newly released Macintosh System 7.5 features. SmalltalkAgents is an interactive, dynamic object-oriented development environment providing powerful application development tools which tightly integrate Macintosh OS functionality

SmalltalkAgents v.1.2 DR/1 supports DoScript AppleEvents, Macintosh Drag & Drop, multi-threaded execution, and the asynchronous SCSI Manager 4.3. Support for AOCE services and QuickTime 2.0 will be included in version 1.2 DR/2.

Technical contact: David Buell, (301) 530-4853, dbuell@qks.com. Marketing contact: Nobuko Isomata, (301) 530-4853, isomata@qks.com

PhoneBridge® Developer's Kit Available

Collaboration Technologies announces the 'Pre-Release Developer Version' of PhoneBridge, the voice telephony platform for all Macintosh software developers. It's a peripheral device that integrates the functions of a desktop telephone and voice mail system into any Macintosh computer.

A few of PhoneBridge's features include: integrated headset, caller ID decode, distinctive ring decode, DTMF encode/decode, record and play audio to and from the phone line, over-the-phone speech recognition, text-to-speech generation, and enhanced call progress detection.

Developers can control all hardware and software features of PhoneBridge, and can create applications that include integration of voice mail, computer-controlled answering/dialing, interactive voice response, and more, into such environments as help desks, telemarketing centers, purchasing departments, and other telephone intensive operations.

PhoneBridge is completely Macintosh Telephony Architecture (MTA) compliant and allows easy integration with existing code, databases, and scripts and is compatible with all popular development environments such as Metrowerks, THINK, MPW, databases, AppleScript, and HyperCard.

The 'Pre-Release Developer Version' is available now, and includes feature-complete prototype hardware and the

Continued on page 86

NewsBits continued from page 85

PhoneBridge SDK (still in development) for $500 plus tax and shipping (where applicable). The kit includes a free upgrade to the final product when available.

The commercial version will ship next quarter and for under $300. Bundled software includes an on-screen dialer and a simple answering machine. The SDK includes the PhoneBridge Telephone Tool, PhoneBridgeIAC (for control via Telephony AppleEvents), and sample code for popular development environments. PhoneBridge is compatible with PowerTalk, AppleEvents, AppleScript, Macintosh Drag and Drop, Telephone Manager, and System 7.5.

To place orders or for more information, contact Wayne D. Correia of Collaboration Technologies at (415) 494-6611 *voice*, (415) 494-6962 *fax* or send e-mail to info@collab.com.

New Jasik Update

Jasik Designs announces an update to The Debugger & MacNosy. Enhancements include:

- Memory Watch for PowerPC
- Directly change the values of variables, registers and memory in the windows they appear in (just like a real Mac program)
- Set/clear breakpoints by clicking in the left edge of a code window (the cursor changes to a 'stop' sign)
- Change window titles by typing into them (Sticky Notes á la Jasik)
- Miscellaneous Human Interface improvements such as a new scroll bar CDEF with double arrows to indicate page scrolling, etc

The update includes many bug fixes and other little changes to make using The Debugger more enjoyable For more info, contact Jasik Designs at macnosy@netcom.com or (415) 322-1386 *voice*.

NeoAccess 3.0

NeoLogic announces NeoAccess 3.0, their cross-platform object database engine. NeoAccess provides object persistence as well as the organizational and fast searching capabilities of a high-end database engine, with a memory footprint of under 100K. New features and enhancements include: multi-threading support with object locking, thread and semaphore classes; the ability to dynamically add and remove indices from a class; consolidated indices that organize all objects having a given base class together; support for multiple blobs or part lists in a single object; improved architecture which reduces the need to subclass; even better performance.

The NeoAccess Developer's Toolkit sells for $749 per developer (plus shipping and tax) with no runtime licensing fees. It includes full source, numerous sample applications, 450+ pages of well-written documentation, and 30 days of online technical support.

For more info, contact NeoLogic Systems, 1450 Fourth St, Suite 12, Berkeley, CA 94710. (510) 524-5897 *voice*, (510) 524-4501 *fax*, neologic@holonet.net, CIS 71762,214, AOL NeoLogic, AppleLink NeoLogic

Dialog Box continued from page 84

thing for illustrations and labeling. But couldn't you refrain from using gradients in the boxes (sunbursts or whatever) at the beginning of each article? They make the colors look dirty and mars the otherwise sober layout of the magazine. Use vivid and pure colors only – it actually improves on the impact the colors have.

But aside from my gradient grudge, I must say that I like your magazine very much. Sprocket is a great idea in this age of hyperbloated fatware (I just got In Control 3.0 – it's grown and become so slow your teeth fall out when you have to wait for the auto-enter function!).

That's all for this instance.

— Piet Seiden, seiden@biobase.aau.dk
Frederiksberg, Denmark

P.S. Dilbert doesn't suck! Keep him around.

Should MacTech Take Sides?

After reading the October Dialog Box letter from Stephen Johnson I felt compelled to respond. Last time I checked, MacTech Magazine was owned by Xplain Corporation, not Apple Computer. As such, there is no reason why MacTech should stick by Apple just because it's Apple. Regardless of what technology is better, Windows does exist and will continue to exist. I'm a realist and because of this I want to know all sides of a technology debate. I would be much less inclined to read MacTech if it always tried to make Apple look good and shelter me from "the bad guys at Microsoft". Apple is not a small child. They can do their own marketing. The day Apple needs to rely on a publication to be unfairly biased to convince Macintosh developers in a technology debate is the day Apple has truly lost the technology war. MacTech is not Stars and Stripes. It's not a "feel good" magazine. MacTech provides information that is important to Macintosh developers, not its opinion of what is best for us. I'm grown up and I can make my own decisions. I don't need a magazine to tell me what technology to use.

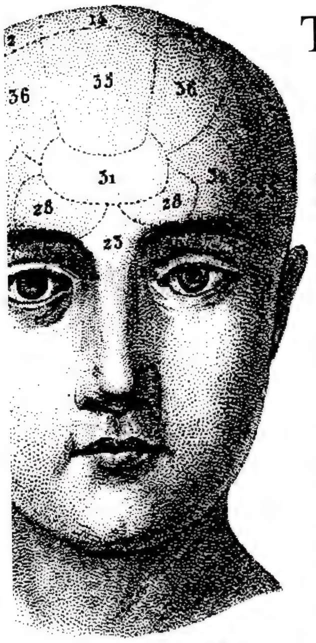
Macintosh enthusiasts seem to forget that throughout the brief life of the computer industry, rarely has the "best" technology become a standard. It's the technology that's "good enough" which emerges from the pack. Living a sheltered life doesn't make bad technology go away – it leaves you unprepared for reality.

I have a very strong emotional attachment to the Macintosh. I've been programming the Macintosh for over 1/3 of my life and I cannot imagine programming any other computer. If Apple bites the dust, I'll find another industry to work in rather than write Windows software. Still, if Apple cannot stand on its own, I'm not willing to live in a cave and pretend it does.

Wake up pal. It's time to face reality.

— Steve Kiene, mindvision@mindvision.com

MACINTOSH SOFTWARE ENGINEERING POSITIONS



We Understand The Programmer's Mind.℠

Bateman Inc. is one of the country's dominant forces in the recruitment of Microsoft Windows, Macintosh and OS/2 developers. Our scope is nationwide, and we service software development firms spanning the spectrum from small, venture capitalized firms to industry giants such as Claris, Microsoft and Borland. We currently seek...

Mac/C++ Developers

Our client is a leader in the development of scalable compression technology, serving the high-growth multimedia markets of on-line services, CD-ROM and video communications. Currently they seek strong Macintosh/C++ developers to work on their retail-level image editing and digital video composition/editing products. 3 years experience, shrink-wrap product background is desired, experience in digital video or imaging applications a plus, but not mandatory. Salary based on experience, California location.

Bateman Inc.
5847A Uplander Way
Culver City, CA 90230
Tel: 310-641-4100 Fax: 310-641-2900

© 1994. "We Understand The Programmer's Mind" is a service mark of Bateman Inc.

MacRegistry™

Developer Job Opportunities

If you are a Macintosh developer, you should register with us! We have a database that enables us to let you know about job opportunities. When we are asked to do a search by a client company the database is the first place we go. There is no charge for registering. The database service is free. Geographic Coverage is nationwide.

Marketability Assessment - To get a specific feel for your marketability send a resumé via Email or call. You may also request a Resume Workbook & Career Planner.

Discreet - We are very careful to protect the confidentiality of a currently employed developer.

Scientific Placement is managed by graduate engineers, we enjoy a reputation for competent & professional job placement services and we are Mac fanatics.

1-800-231-5920 | AppleLink: D1580 | 713-496-0373 fax

Scientific Placement, Inc.

Dept. MT MacRegistry P.O. Box 19949 Houston, Texas 77224 (713) 496-6100 das@scientific.com	Dept. MT MacRegistry P.O. Box 71 San Ramon, CA 94583 510-733-6168 bge@scientific.com	Dept. MT MacRegistry P.O. Box 4270 Johnson City, TN 37602 (615) 854-9444 rjg@scientific.com

Do you have a service...

that you want to advertise to over 30,000 Macintosh programmers and developers?

Do it in the MacTech Magazine Classifieds – an inexpensive vehicle that *gets results.*

Call Ruth Subrin at 310/575-4343 to reserve space NOW!

MacTech Magazine is your recruitment vehicle

When you need to fill important positions at your company, MacTech Magazine is the consistent choice of companies across the country for hiring the best qualified Macintosh programmers and developers. Let MacTech Magazine deliver your recruitment message to an audience of over 27,000 qualified computer professionals.

Call Ruth Subrin at 310/575-4343

WANTED

Dead or Alive

Dead or Alive

MacApp Programmers

MacXperts is a software development company which has immediate openings for people who are experienced MacApp programmers and for C++ pro-grammers who want to learn MacApp or NTK. If you have what it takes, and the desire to achieve, call Kendall Tyler at MacXperts.

Voice: 800-356-8040 Fax: 804-358-3847
Internet: xperts@infi.net
AppleLink: xperts AOL: Xperts

If you have a CD-ROM drive, then you've *gotta have* every article published in the first 8 years of MacTech Magazine, including all of 1992. Over 900 articles. All 95 issues. All the source code. THINK Reference 2.0 On Location 2.0 Working applications. Full documentation. Demos for developers. Free shareware code.

Plus, Plus, Plus...

FREE Upgrade to next version of CD-ROM!!*

*Any registered user who purchases *All of MacTech Magazine CD-ROM Volume 1-8* after November 1, 1993 will receive a free upgrade to the next version of the All of MacTech Magazine CD-ROM when available.

More than 500 megs, selected by Macintosh developers for Macintosh developers, all on the new All of MacTech Magazine CD-ROM, Volume 1–8.

In The MacTech Magazine Folder

- Every article published in the first 8 years
- Complete source code & working examples: Assembly, Pascal, C, FORTRAN, BASIC, Lisp, MacScheme, Forth, Neon, MOPS, Yerk, Prograph, Hypercard, Ada, 4th Dimension, MacApp, THINK Class Library
- Complete text of articles including dialogs, custom windows and menus, sounds, viruses, object oriented programming, animation, simulations, XCMDs and XFCNs, parsing, User Interface Design, List Manager, Class Libraries, debugging, product reviews, inits, control panel devices.
- Special ViewR document reader to view, search and print articles.
- And much more!

THINK™ Reference

FREE!

Symantec's THINK Reference 2.0. Complete on line guide to Inside Macintosh, Vol. I-VI, with cross referenced index, detailed information of each function, procedure and detail needed when programming the Macintosh.

SYMANTEC.™

In The Apple Folder

- "Up Your FCBs"
- BitMapToRegion
- CD-ROM 3.2
- Discipline
- Exec
- MacsBug 6.2.2
- Basic Connectivity Set w/Communications Toolbox
- System 7.0.1 with Tune-up
- Logic Manager 1.0d2
- ResEdit 2.1.1+
- QuickTime 1.5
- And much more!

FREE! ON LOCATION™

A full working, registered version of On Location 2.0, from On Technology. Use it to index all your source code disks. Find references in seconds on multiple disks whether or not the disk is mounted.

Demos relevant to developers

- Demos for all kinds of third party tools and utilities

Public/Shareware Code

- 150+ meg, most are fully documented, including many updates
- Best of BMUG's Programming Tools, System 7 Utilities, etc.
- Hundreds of code ideas, explanations, help files
- Tutorials
- Source code for more than 70 working applications
- And much, much more!

More! More! More!

We could go on listing everything that is included on this new exciting CD-ROM. But you should see it for yourself. Order yours today, for $299 (upgrades for $150) plus shipping & handling. Call, fax, or e-mail for more information.

MacTech MAGAZINE™ Formerly MacTutor

FOR MACINTOSH PROGRAMMERS & DEVELOPERS

Voice: 310/575-4343 • Fax: 310/575-0925
AppleLink: MT.CUSTSVC • CompuServe: 71333,1063
Internet: custservice@xplain.com
America Online & GEnie: MACTECHMAG

MACTECH MAGAZINE PRODUCTS & ORDER INFORMATION

E-mail, Fax, write, or call us. You may use your VISA, MasterCard or American Express; or you may send check or money order (in US funds only): MacTech Magazine, P.O. Box 250055, Los Angeles, CA 90025-9555.
Voice: 310/575-4343 • Fax: 310/575-0925

If you are an e-mail user, you can place orders or contact customer service at:
- **AppleLink:** MT.CUSTSVC
- **CompuServe:** 71333,1063
- **Internet:** custservice@xplain.com
- **America Online:** MT CUSTSVC
- **GEnie:** MACTECHMAG

SUBSCRIPTIONS
US Magazine: $47 for 12 issues
Canada: $59 for 12 issues
International: $97 for 12 issues
Domestic source code disk: $77 for 12 issues
Int'l source code disk: $97 for 12 issues

CD-ROM
All of MacTech Volumes I-VIII CD-ROM: Includes over 900 articles from all 95 issues (1984-1992) of MacTech Magazine (formerly MacTutor). All article text and source code. Articles and code are indexed by On Location. The CD includes Symantec's THINK™ Reference 2.0, On Technology's On Location™ 2.0, working applications with full documentation, product demos for developers and more. See advertisement, this issue: $299. Upgrades $150, e-mail, call or write for info.

BOOKS
The Best of MacTutor, Volume 1: **Sold Out**
The Complete MacTutor, Volume 2: **Sold Out**
The Essential MacTutor, Volume 3: $19.95
The Definitive MacTutor, Volume 4: $24.95
The Best of MacTutor, Volume 5: $34.95
Best of MacTutor Collection, Volumes 3 – 5: $69
Best of MacTutor, Volumes 6, 7, 8 & 9: Not available

DISKS
Source Code Disks: $8 each
Topical Index (1984-1991) on disk: $5

MAGAZINE BACK ISSUES
Volumes 3, 4, 5, 6, 7, 8, 9 and 10: $5 each (subject to availability)

SHIPPING, HANDLING & TAXES
California:
Source disk or single issue: $3
Single book or multiple back issues: $5
Two books: $8 • All other orders: $12

California residents include 8.25% sales tax on all software, disks and books.

Continental US:
Source disk or single issue: $3
Single book or multiple back issues: $7
Two books: $15 • All other orders: $17

Canada, Mexico and Overseas: Please contact us for shipping information.

Allow up to 2 weeks for standard domestic orders, more time for international orders.

PLEASE NOTE
Source code disks and journals from MacTech Magazine are licensed to the purchaser for private use only and are not to be copied for commercial gain. However, the code contained therein may be included, if properly acknowledged, in commercial products at no additional charge. All prices are subject to change without notice.

3RD PARTY PRODUCTS

10% OFF ALL BOOKS!

MACTECH EXCLUSIVES

MacTech Magazine is your exclusive source for these specific products:

NEW! **Ad Lib 2.0** The premier MacApp 3.0 compatible ViewEdit replacement. A powerful user-interface editing tool to build views for MacApp 3.0 and 3.1. Ad Lib allows subclassing of all of MacApp's view classes including adorners, behaviors, and drawing environments. String and text style resources are managed automatically. Alternate display methods, such as a view hierarchy window, allow easy examination of complex view structures. Ad Lib includes source code for MacApp extensions that are supported by the editor – buttons can be activated by keystrokes, behaviors can be attached to the application object, and general purpose behaviors can be configured to perform a number of useful functions. Run mode allows the user to try out the views as they will work in an application. Templates can be created to add additional data fields to view classes. Editing palettes provide fast and easy editing of common objects and attributes. Works with ACI's Object Master (version 2.0 and later) to navigate a project's user interface source code. $195

NEW! **FrameWorks Magazine:** $8/issue, subject to availability.

NEW! **FrameWorks Source Code Disk:** $10/issue, subject to availability.

NEW! **Five Years of Objects CD-ROM:** FrameWorks archives and source code from April 1991 to January 1993, plus selected object-oriented publicly available software and demos. $95

MADACON '93 CD-ROM: The highlights of MADACON '93, including Mike Potel on Pink, Bedrock, MacApp, OODLs, and more. Slides, articles, demos, audio, and QuickTime. $95

NEW! **MAScript 1.2** adds support for AppleScript to your MacApp 3.0.1 and 3.1 based applications. Make your application scriptable and recordable by building on a tried and tested framework for object model support. MAScript dispatches Apple events to the appropriate objects, creates object specifiers, and makes framework objects like windows and documents scriptable and recordable. Sample application shows you how to begin adding support for scripting and recording. MAScript includes complete source code. Install MAScript by modifying one MacApp source file, then adding another to your project. Future versions of MacApp will incorporate MAScript, so MAScript support you add now will work in the future. $199

NEW! **The Mjølner BETA System** is a software development environment supporting object-oriented programming in the BETA programming language. BETA is uniquely expressive and orthogonal. BETA unifies just about every abstraction mechanism - including class, procedure, function, coroutine, process and exception - into the ultimate abstraction mechanism: the pattern. BETA includes: general block structure, strong typing, whole/part objects. The compiler: binary code generation, automatic garbage collection, separate compilation, interface to C, Pascal, and assembler.

The system: persistent objects, basic libraries with containers classes, platform-independent GUI application frameworks on Unix, Mac and Windows NT, metaprogramming system. The tools available on Unix: the hyper structure editor supporting syntax directed editing, browsing, etc., and the source code debugger are currently being ported to the Macintosh system. The Mjølner BETA System for Macintosh requires MPW (basic set) 3.2 or later.

Package containing compiler, basic libraries, persistent store, GUI framework, and comprehensive documentation. (Other packages are also available) $295

NEW! **Savvy** OSA support includes attachability, recordability, scriptability, coercion, in addition to script execution, idling and i/o. Apple event support includes complex object specifiers, synchronous/asynchronous Apple event handling, and Apple event transactions for clients and servers. The Core Suite of Apple event objects is supporting including the application, documents, windows, and files. Documentation includes technology overview, cookbook, and sample code. $250

NEW! **More Savvy** includes all Savvy features plus Apple event support for all sub-classes of TEventHandler with extensive view support. Apple event support for text includes text attributes and sub-range specification. Recordability supports additional actions, and coercion includes additional types. Additional client and server Apple events. $450

NEW! **Super Savvy** includes all More Savvy features plus compile, edit, and record scripts using built in script editor. View template editors, like Ad Lib, can attach scripts to view objects and modified scripts are saved with the document. Script action behavior allow quick access for executing and editing scripts attached to views. Text to object specifier coercion plus more. $700

NEW! **Savvy QuickTime** Requires Savvy, More Savvy, or Super Savvy. Includes QuickTime, Apple event and view template support. Movies come out of the box ready to play, edit, and react to Apple events. They can be included in any view structure, including templates, and are displayed in the scrap view. Movie controls include volume, play rate, looping mode, display style, and other characteristics. $250

NEW! **Savvy DataBase** Requires Savvy, More Savvy, or Super Savvy. Available Winter 1994. $250

MacTech Magazine is your exclusive source for available back issues of SFA's magazine, source code disks and assorted CD's. Call for more info and pricing.

BOOKS

Defying Gravity: The Making of Newton Doug Menuez and Markos Kounalakis. An in depth, dramatic account of the story of Newton's creation. It is a techno-logical adventure story; a fascinating case study of the process by which an idea is born and then translated into a product on which careers and fortunes can be made or lost. It is a new kind of business book, one that captures through powerful photo-journalism and a fast-paced text, the human drama and risk involved in the invention of a new technology for a new marketplace. 196 pgs., ~~$29.95~~ **$26.95**

The Elements of E-Mail Style by Brent Heslop and David Angell. Learn the rules of the road in the e-mail age. Concise, easy-to-use format explaining essential e-mail guidelines and rules. It covers style, tone, typography, formatting, politics and etiquette. It also outlines basic rules of composition within the special context of writing e-mail and includes samples and templates for writing specific types of e-mail correspondence. 208 pages. ~~$14.95~~ **$13.45**

NEW! **E-Mail Essentials** by Ed Tittel & Margaret Robbins is a hands-on guide to the basics of e-mail, the ubiquitous networks communication system. The book is suitable for both the casual e-mailer and the networking professional, as it covers everything from the installation of e-mail to the maintenance and management of e-mail hubs and message servers. The books explains the fundamental concepts and technologies of electronic mail, featuring chapters on Lotus applications and CompuServe, as well as information on upgrading, automation, message-based applications, and user training. E-mail is a step-by-step, jargon-free guide that will enable the e-mail user to get the most out of the communication potentials of networking. 250 pp. ~~$24.95~~ **$22.45**

NEW! **Graphics Gems IV** edited by Paul Heckbert Volume IV is the newest collection of carefully crafted, innovative gems. All of the gems are immediately accessible and useful in formulating clean, fast, and elegant programs. The C programming language is used for most of the program listings, although several of the gems have C++ implementations. An IBM or Macintosh disk containing all of the code from all four volumes is included. Includes one 3.5" high-density disk. ~~$49.95~~ **$44.95**

How To Write Macintosh Software by Scott Knaster is a great source for understanding Macintosh programming techniques. Drawing from his years of experience working with programmers, Scott explains the mysteries and myths of Macintosh programming with wit and humor. The third edition, fully revised and updated, covers System 7 and 32-bit developments, and explores such topics as how and where things are stored in memory; what things in memory can be moved around and when they may be moved; how to debug your applications with MacsBug; how to examine your program's code to learn precisely what's going on when it runs. 448 pgs., ~~$28.95~~ **$26.05**

The Instant Internet Guide by Brent Heslop and David Angell. An Internet jump-start – how to access, use and navigate global networks. The Instant Internet Guide equips readers with the tools needed to travel the electronic world. The book highlights the most important sources of Internet news and information and explains how to access information on remote systems. It outlines how to use essential Internet utilities and programs and includes a primer on UNIX for the Internet. 224 pages ~~$14.95~~ **$13.45**

Learn C on the Macintosh by Dave Mark. This self-teaching book/disk package gives you everything you need to begin programming on the Macintosh. Learn to write, edit, compile, and run your first C programs through a series of over 25 projects that build on one another. The book comes with THIN C – a customized version of Symantec's THINK C, the leading programming environment for Macintosh. 464 pages, Book/disk: ~~$34.95~~ **$31.45**

Learn C++ on the Macintosh by Dave Mark. After a brief refresher course in C, Learn C++ introduces the basic syntax of C++ and object programming. Then you'll learn how to write, edit, and compile your first C++ programs through a series of programming projects that build on one another as new concepts are introduced. Key C++ concepts such as derived classes, operator overloading, and iostream functions are all covered in Dave's easy-to-follow approach. Includes a special version of Symantec C++ for Macintosh. Book/disk package with 3.5" 800K Macintosh disk. 400 pages, ~~$36.95~~ **$33.26**

Macintosh C Programming Primer Volume I, Second Edition, Inside the Toolbox Using THINK C by Dave Mark and Cartwright Reed. This new edition of this Macintosh programming bestseller is updated to include recent changes in Macintosh technology, including System 7, new versions of THINK C and ResEdit, and new Macintosh machines. Readers will learn how to use the resources, Macintosh Toolbox and interface to create stand-alone applications. 672 pages, ~~$26.95~~ **$24.25**

Macintosh C Programming Primer Volume II, Mastering the Toolbox Using THINK C by Dave Mark. Volume II picks up where Volume I leaves off, covering more advanced topics such as: Color QuickDraw, THINK Class Library, TextEdit, and the Memory Manager: 528 pgs. ~~$26.95~~ **$24.25**

Macintosh Pascal Programming Primer Volume I, Inside the Toolbox Using THINK Pascal by Dave Mark and Cartwright Reed. This tutorial shows programmers new to the Macintosh how to use the Toolbox, resources, and the Macintosh interface to create stand-alone applications with Symantec's THINK Pascal. 544 pages ~~$26.95~~ **$24.25**

Macintosh Programming Techniques by Dan Sydow (Series Editor: Tony Meadow). This tutorial and handbook provides a thorough foundation in the special techniques of Macintosh program-ming for experienced Macintosh programmers as well as those making the transition from DOS, Windows, VAX or UNIX. Emphasizes programming techniques over syntax for better code, regardless of language. Guides the reader through Macintosh memory management, QuickDraw, events and more, using sample program in C++. Disk includes an interactive tutorial, plus reusable C++ code. ~~$34.95~~ **$31.95**

NEW! **Multimedia Authoring Building and Developing Documents** by Scott Fisher addresses the concerns that face anyone trying to create multimedia documents. It offers specific advice on when to use different kinds of information architecture, discusses the human-factors concepts that determine how readers use and retain information, and them applies these findings to multimedia documents, covering the high-level issues concerning planners and authors of multimedia documents as well as those involved in evaluating or purchasing multimedia platforms. Includes one 3.5" high-density disk. ~~$34.95~~ **$31.45**

NEW! **Programming for the Newton Software Development with NewtonScript** by Julie McKeehan and Neil Rhodes. Foreword by Walter R. Smith. Programming for the Newton: Software Development with NewtonScript is an indispensable tool for Newton programmers. Readers will learn how to develop software for the Newton on the Macintosh from people that developed the course on programming the Newton for Apple Computer. The enclosed 3.5" disk contains a sample Newton application from the books, as well as demonstration version of Newton Toolkit (NTK), Apple Computers complete development environment for the Newtons. A Publication of AP Professional May 1994, Paperback, 393 pp. ~~$29.95~~ **$26.95**

Programming in Symantec C++ for the Macintosh by Judy May and John Whittle. This book will introduce you to object-oriented programming, the C++ language, and of course Symantec C++ for the Macintosh. You don't have to be a programmer, or even know anything about programming to benefit from this book. Programming in Symantec C++ for the Macintosh covers everything from the basics to advanced features of Symantec C++. If you are a Think C or Zortech C++ programmer who wants to learn more about object-oriented programming or what's different about Symantec C++, there are whole chapters specifically for you. Includes helpful examples of C++ code that illustrate object-oriented programs. ~~$29.95~~ **$26.95**

Programming for System 7 by Gary Little and Tim Swihart, is a hands-on guide to creating applications for System 7. It describes the new features and functions of the operating system in detail. Topics covered include file operations, cooperative multitasking, Balloon Help, Apple events, and the File Manager. Numerous working C code examples show programmers how to take advantage of each of these features and use them in developing their applications. 384 pages ~~$26.95~~ **$24.25**

ResEdit™ Complete, Second Edition by Peter Alley and Carolyn Strange. With ResEdit, Macintosh programmers can customize every aspect of their interface

Want more product info? Call us at 310/575-4343.

form creating screen backgrounds and icons to customizing menus and dialog boxes. 608 pages. Book/disk package. ~~$34.95~~ **$31.45**

Sad Macs, Bombs, Disasters and What to Do About Them by Ted Landau comes to the rescue with your Macintosh problems. From fractious fonts to the ominous Sad Macintosh icon, this emergency handbook covers the whole range of Macintosh problems: symptoms, causes, and what you can do to solve them. 640 Pages ~~$24.95~~ **$22.45**

Software By Design: Creating User Friendly Software by Penny Bauersfeld (Series Editor: Tony Meadow). This excellent reference provides readers with a thorough how-to for designing software that is easy to learn, comfortable to operate and that inspires user confidence. Written from the perspective of Macintosh, but compatible with all platforms. Stresses user input from initial design, through prototyping, testing and revision. Provides tools for analyzing user needs and test responses. Includes exercises for sharpening user-oriented design skills. ~~$29.95~~ **$26.95**

NEW! **Taligent's Guide to Designing Programs Well-Mannered Object-Oriented Design in C++** is the Taligent approach to object-oriented design The Taligent Operating Environment is the first commercial software system based entirely on object-oriented technology. Taligent's Guide to Designing Programs is a developer's-eye view of this system. It introduces new concepts of programming and empowers developers to create software more productively. Out of their direct experience in developing the system, the authors focus on global issues of object-oriented design and writing C++ programs, and the specific issues of programming in the Taligent Operating Environment. Taligent's Guide to Designing Programs assumes the reader is an experienced C++ programmer, and proceeds from there to fully explore "the Taligent way" of programming. ~~$19.50~~ **$17.55**

Writing Localizable Software for the Macintosh by Daniel F. Carter. 469 pages. ~~$26.95~~ **$24.25**

THE APPLE LIBRARY

HyperCard Stack Design Guidelines by Apple Computer, Inc. is an essential book for everyone who creates Apple HyperCard stacks, from beginners to commercial developers. It covers the basic principles of design that, when incorporated, make HyperCard stacks effective and usable. Topics include guidelines, navigation, graphic design and screen illustration, text in stacks, music and sound, a sample stack development scenario, collaborative development, and the Stack Design Checklist. 240 pages, ~~$21.95~~ **$19.95**

Inside AppleTalk by Gursharan S. Sidhu, Richard F. Andrews and Alan B. Oppenheimer. Apple Computer, Inc. 650 pages, ~~$34.95~~ **$31.45**

Inside Macintosh: AOCE Application Interfaces by Apple Computer, Inc. shows how your application can take advantage of the system software features provided by PowerTalk system software and the PowerShare collaboration servers. Nearly every Macintosh application program can benefit from the addition of some of these features. This book shows how you can add electronic mail capabilities to your application, write a messaging application or agent, store information in and retrieve information from PowerShare and other AOCE catalogs, add catalog-browsing and find-in-catalog capabilities to your application, write templates that extend the Finder's ability to display information in PowerShare and other AOCE catalogs, add digital signatures to files or to any portion of a document, and establish an authenticated messaging connection. ~~$40.45~~ **$36.40**

Inside Macintosh: AOCE Service Access Modules by Apple Computer, Inc. describes how to write a software module that gives users and PowerTalk-enabled applications access to a new or existing mail and messaging service or catalog service. This book shows how to write a catalog service access module (CSAM), a messaging service access module (MSAM), and AOCE templates that allow a user to set up a CSAM or MSAM and add addresses to mail and messages. ~~$26.95~~ **$24.25**

NEW! **Inside Macintosh: CD-ROM** by Apple Computer, Inc. is the essential guide to using AppleScript to customize and automate the Macintosh. Inside Macintosh® is the essential reference for programmers, designers, and engineers for creating applications for the Macintosh family of computers. Inside Macintosh CD-ROM collects more than 25 volumes in electronic form, including: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components. Now programmers will be able to access over 16,000 pages of the information they need directly from their computers. Hypertext linking and extensive cross referencing across volumes allows programmers to search and explore this library in ways that are unique to the electronic medium. Every Macintosh programmer will regard Inside Macintosh CD-ROM as their most important resource. $99.95

Inside Macintosh: Devices by Apple Computer, Inc. describes how to write software that interacts with built-in and peripheral hardware devices. With this book,. you'll learn how to write and install your own device drivers, desk accessories, and Chooser extensions; communicate with device drivers using the Device Manager; access expansion cards using the Slot Manager; control SCSI devices using SCSI Manager 4.3 or the original SCSI Manager; communicate directly with Apple Desktop Bus devices; interact with the Power Manager in battery-powered Macintosh computers; and communicate with serial devices using the Serial Driver. ~~$29.95~~ **$26.95**

Inside Macintosh: Files by Apple Computer, Inc. describes the parts of the operating system that allow you to manage files. It shows how your application can handle the commands typically found in a File menu. It also provides a reference to the File and Alias Managers, the Disk Initialization and Standard File Packages. 510 pgs, ~~$29.95~~ **$26.95**

Inside Macintosh: Interapplication Communication by Apple Computer, Inc. shows how applications can work together. How your application can share data, request information or services, allow the user to automate tasks, communicate with remote databases. ~~$34.95~~ **$31.45**

Inside Macintosh: Imaging by Apple Computer, Inc. covers QuickDraw and Color QuickDraw. The book includes general discussions of drawing and working with color. It describes the structures that hold images and image information, and the routines that manipulate them. It also covers the Palette, Color, and Printing Managers, and the Color Picker, Color Matching, and Picture Utilities. ~~$26.95~~ **$24.25**

Inside Macintosh: Macintosh Toolbox Essentials by Apple Computer, Inc. covers the heart of the Macintosh. The toolbox enables programmers to create applications consistent with the Macintosh "look and feel". This book describes Toolbox routines and shows how to implement essential user interface elements, such as menus, windows, scroll bars, icons and dialog boxes. 880 pages ~~$34.95~~ **$31.45**

Inside Macintosh: More Macintosh Toolbox by Apple Computer, Inc. covers other Macintosh features such as how to support copy and paste, provide Balloon Help, play and record sound and create control panels are covered in this volume. The managers discussed include Help, List, Resource, Scrap and Sound. ~~$34.65~~ **$31.45**

Inside Macintosh: Memory by Apple Computer, Inc. describes the parts of the Macintosh operating system that allow you to manage memory. It provides detailed strategies for allocating and releasing memory, avoiding low-memory situations, reference to the Memory Manager, the Virtual Memory Manager, and memory-related utilities. 296 pages, ~~$24.95~~ **$22.45**

Inside Macintosh: Networking by Apple Computer, Inc. describes how to write software that uses AppleTalk networking protocols. It describes the components and organization of AppleTalk and how to select an AppleTalk protocol. It provides the complete application interfaces to all AppleTalk protocols, including ATP (AppleTalk Transaction Protocol), DDP (Datagram Delivery Protocol), and ADSP (AppleTalk Data Stream Protocol), among others. ~~$29.95~~ **$26.95**

Inside Macintosh: Operating System Utilities by Apple Computer, Inc. describes parts of the Macintosh Operating System that allow you to manage various low-level aspects of the operating system. Everyone who programs the Macintosh should read this book! It will show you in detail how to get information about the operating system, manage operating system queues, handle dates and times, control the settings of the parameter RAM, manipulate the trap dispatch table, and receive and respond to low-level system errors. ~~$26.05~~ **$23.45**

Inside Macintosh: Overview by Apple Computer, Inc. is the first book that people who are unfamiliar with Macintosh programming should read. It gives an overview of Macintosh programming fundamentals and a road map to the New Inside Macintosh library. Inside Macintosh: Overview also covers various programming tools and languages, compatibility guidelines and an overview of considerations for worldwide development. 176 pages, ~~$22.95~~ **$20.65**

Inside Macintosh: PowerPC Numerics by Apple Computer, Inc. describes the floating-point numerics environment provided with the first release of PowerPC processor-based Macintosh computers. The numerics environment conforms to the IEEE standard 754 for binary floating-point arithmetic. This book provides a description of that standard and shows how RISC Numerics compiles with it. This book also shows programmers how to create floating-point values and how to perform operations on floating-point values in high-level languages such as C and in PowerPC assembly language. ~~$28.95~~ **$26.00**

Inside Macintosh: PowerPC System Software by Apple Computer, Inc. describes the new process execution environment and system software

Want more product info? E-mail us at productinfo@xplain.com

services provided with the first version of the system software for Macintosh on PowerPC computers. It contains information you need to know to write applications and other software that can run on the PowerPC. PowerPC System Software shows in detail how to make your software compatible with the new run-time environment provided on PowerPC-based Macintosh computers. It also provides a complete technical reference for the Mixed Mode Manager, the Code Fragment Manager, and the Exception Manager. ~~$24.95~~ **$22.45**

Inside Macintosh: Processes by Apple Computer, Inc. describes the parts of the Macintosh operating system that allow you to control the execution of processes and interrupt tasks. It shows in detail how you can use the Process Manager to get information about processes loaded in memory. It is also a reference for the Vertical Retrace, Time, Notification, Deferred Task, and Shutdown Managers. 208 pages, ~~$22.95~~ **$20.65**

Inside Macintosh: QuickTime by Apple Computer, Inc. is for anyone who wants to create applications that use QuickTime, the system software that allows the integration of video, animation, and sounds into applications. This book describes all of the QuickTime Toolbox utilities. In addition, it provides the information you need to compress and decompress images and image sequences. ~~$29.95~~ **$26.95**

Inside Macintosh: QuickTime Components by Apple Computer, Inc.covers how to use and develop QuickTime components such as image compressors, movie controllers, sequence grabbers, and video digitizers. ~~$34.95~~ **$31.45**

Inside Macintosh: Sound by Apple Computer, Inc. describes the parts of the Macintosh system software that allow you to manage sounds. It contains information that you need to know to write applications and other software that can record and play back sounds, compress and expand audio data, convert text to speech, and perform other similar operations. ~~$26.95~~ **$24.25**

Inside Macintosh: Text by Apple Computer, Inc. describes how to perform text handling, from simple character display to multi-language processing. The Font, Script, Text Services, and Dictionary Managers are all covered, in addition to QuickDraw Text, TextEdit, and International and Keyboard Resources. ~~$39.95~~ **$35.95**

Inside Macintosh: QuickDraw™ GX Library by Apple Computer, Inc. is the powerful new graphics architecture for the Macintosh. Far more than just a revision of QuickDraw, QuickDraw GX is a unified approach to graphics and typography that gives programmers unprecedented flexibility and power in drawing and printing all kinds of shapes, images, and text. This long-awaited extension to Macintosh system software is documented in a library of books that are themselves an extension to the new Inside Macintosh series. The QuickDraw GX Library is clear, concise, and organized by topic. The books contain detailed explanations and abundant programming examples. With extensive cross-references, illustrations, and C-language sample code, the QuickDraw GX Library gives programmers fast and complete reference information for creating powerful graphics and publishing applications with sophisticated printing capabilities. The first two volumes in the QuickDraw GX Library are:

Inside Macintosh: QuickDraw GX Objects by Apple Computer, Inc. introduces QuickDraw GX and its object structure, and shows programmers how to manipulate objects in all types of programs. ~~$26.95~~ **$24.25**

Inside Macintosh: QuickDraw GX Graphics by Apple Computer, Inc. shows readers how to create and manipulate the fundamental geometric shapes of QuickDraw GX to generate a vast range of graphic entities. It also demonstrates how to work with bitmaps and pictures, and specialized QuickDraw GX graphic shapes. ~~$26.95~~ **$24.25**

NEW! **Inside Macintosh:** X-Ref. by Apple Computer, Inc. is an index for Inside Mac. ~~$12.95~~ **$11.65**

LANGUAGES

metrowerks

CodeWarrior™ CD by Metrowerks comes intwo versions – Bronze and Gold. These CDs contain the CodeWarrior development environment including C++, C and Pascal compilers; high-speed linkers; native-mode interactive debuggers; and a powerful new application framework called PowerPlant for rapid Macintosh development in C++. Bronze generates 680x0 code. Gold generates both 680x0 and PowerPC code. All versions are a 3 CD subscription over a 1-year period. Bronze: $99, Gold: $399. **Bronze comes with a 6-month MacTech subscription. Gold comes with a 1-year subscription. Both at no additional charge!**

geekware

NEW! **Geekware** by Metrowerks is here! In high school, they called you a computer geek. Now, they work at burger joints and wear polyester uniforms. And you don't. Wear it to your favorite burger joint. $24.95

FORTRAN by Language Systems is a full-featured ANSI standard FORTRAN 77 compiler that runs in the Macintosh Programmers Workshop (MPW). All major VAX extensions are supported as well as all major features of Cray and Data General FORTRAN. FORTRAN creates System 7 savvy applications quickly and easily. Compiler options specify code generation and optimization for all Macintoshes, including special optimizations for 68040 machines. Error messages are written in plain English and are automatically linked to the source file. The runtime user interface of compiled FORTRAN programs is fully customizable by programmers with any level of Macintosh experience. $595. w/o MPW: $495. Corporate 5 pack $1575

FORTRAN 77 SDK for Power Macintosh by Absoft includes a globally optimizing native compiler and linker, native Fx™ multi-language debugger, and Apple's MPW development environment. The compiler is a full ANSI/ISO FORTRAN 77 implementation and includes all MIL-STD 1753 extensions, Cray/Sun-style POINTER, and several Fortran 90 enhancements. MRWE, Absoft's application framework libraries, is included as is the MIG graphics library for quick creation of plots and graphs. The native Macintosh PPC toolbox is fully supported. Absoft's Fx debugger can debug intermixed FORTRAN 77,C, C++, PPC assembler. The compiler, linker, and debugger all run as native PPC tools and produce native Macintosh PPC executables. $699

MacFortran® II V3.3 is a VAX/VMS compatible, full ANSI/ISO FORTRAN 77 compiler including all MIL-STD 1753 extensions. Acknowledged to be the fastest FORTRAN available for Macintosh, MacFortran II is bundled with the latest version of Macintosh Programmer's Workshop (MPW), and includes SourceBug (Apple's source level symbolic debugger) and SoftwareFPU (a math co-processor emulator). Also included is Absoft's Macintosh Runtime Window Environment (MRWE) application framework (with fully documented source code as examples) and MIG graphics library. MacFortran II v3.3 features improved 68040CPU support and is fully compatible with Power Macintosh under emulation. Documentation includes special sections devoted to use of MacFortran II with the MPW editor and linker, implementation of System 7 features, and porting code to the Macintosh from from various mainframes and Unix workstation platforms. $595

REM
GOSUB
LET
NS BASIC

BASIC for the Newton is BASIC for the Newton! From NS BASIC Corporation, it is a fully interactive implementation of the BASIC programming language. It runs entirely on the Newton - no host is required. It includes a full set of functions and data types, hand-written input, windows, buttons and extensions to take advantage of the Newton environment. Applications can create files or access the built-in soups. Applications can also access the serial port for input and output. Work directly on the Newton, or through a connected Mac/PC and keyboard. NS BASIC includes a 150 page pocket sized manual. $99

QKS

SmalltalkAgents™, a superset of the Smalltalk language, is fully integrated with Macintosh, incorporating design features specifically for the RISC and Macintosh System 7 architecture. SmalltalkAgents is a true object oriented workbench that includes an incremental and extensible compiler, an array of design and cross reference tools, pre-emptive interrupt driven threads and events, an extensive class library including classes for general programming, classes for the Macintosh user interface and classes for the Macintosh operating system. Integration of components in enterprise systems is simplified with the network, telecommunication, and inter-application communication libraries. The SmalltalkAgents' extensive class library and add-on components make it especially well suited as a development workbench for custom applications in business, education, science, engineering, and academic research. $695

SYMANTEC.™

Symantec C++ for Macintosh is an object oriented development environment designed for professional Macintosh programmers. Symantec C++

Want more product info? Call us at 310/575-4343.

features powerful object-oriented development tools within a completely integrated environment. The C++ compiler, incremental linker, THINK Class Library, integrated browser, and automatic project management give Symantec C++ fast turnaround times. This product supports multiple editors and translators, so you can use your favorite tools and resource editors as well as scripts you've written within the environment. And with ToolServer, you'll be able to customize menus and attach scripts based on Apple events, AppleScript, and MPW Tools. The built-in SourceServer provides a source code control system, allowing teams of programmers to solve tough problems faster. With SourceServer, you'll always know you're working on the latest version. And you'll have old versions at your fingertips when code "breaks" and you need to look back at modifications. Product Contents: Three high density disks, an 832-page user manual, a 568-page THINK Class Library and a 100-page C++ Compiler Guide. $369

THINK C by Symantec Corporation. THINK C is easy to use and highly visual, making it the No. 1 selling Macintosh programming environment. Enhancements make this product faster and more versatile than ever, improving your productivity with more powerful project management, a full set of tools, and script support for major script-based languages. With the THINK environment, you spend less time on routine programming tasks due to an extremely fast compiler and incremental linker. In addition, the automatic project manager saves you time by tracking changes to your files and recompiling only those that have changes. All the tools you need – a multi-window editor, compiler, linker, debugger, browser, and resource editor – are completely integrated for speed and ease of use. One of the most valuable of these tools is the THINK Class Library, a set of program building blocks that gives you a head start in writing object-oriented applications. And with the new open architecture, you can use your favorite tools, resource editors, and scripts within the environment. THINK C is the logical next step for programmers who have worked in HyperCard or other script-based development environments. The environment supports AppleScript, Apple events, and Frontier, so you can link and automate complex, multi-project operations. Product Contents: Four Macintosh disks, an 832-page user manual, and a 568-page THINK Class Library Guide. $219

THINK Pascal v. 4.0 by Symantec Corporation. Professionals and students will welcome this version of THINK Pascal. It is fully integrated for rapid turnaround time and lets you take advantage of System 7 capabilities. Features include support for large projects, enhanced THINK Class Library, System 7 compatibility, superior code generation, and smart linking. Product Contents: Four Macintosh disks, a 562-page user manual, and a 498-page object-oriented programming manual. $169

UTILITIES

BBEdit 3.0 from Bare Bones Software is now Accelerated for Power Macintosh. This powerful, intuitive text editor offers integrated support for THINK C 7.0, THINK Reference 2.0 and MPW ToolServer. BBEdit's many features include: Integrated PopUpFuncs™ technology for speedy navigation of source code files, unique "Find Differences" command (BBEdit can find differences between projects and folders as well as files), support for Macintosh Drag and Drop for editing and other common tasks, PowerTalk support for reading, sending and composition of PowerTalk mail, scripting via any OSA compatible scripting language including AppleScript and Frontier 3.0, and fast search and replace with optional "grep" matching and multi-file searching. BBEdit's robust feature set and proven performance and reliability make it the editor of choice for professionals and hobbyists alike. $99

C Programmer's Toolbox/MPW Rev. 3.0 by MMCAD. The C Programmer's Toolbox provides a wealth of programming and documentation support tools for developers who are creating new code, porting existing code, or trying to improve and expand existing code. The tools include: CDecl composes and translates C/C++ declaration statements to/from English; CFlow™ determines program function hierarchy, runtime library contents, function/file interdependencies and graphs all or part of a program's functional structure; CHilite™ highlights and prints C/C++ files; CLint™ semantically checks multiple C source files, identifying potential programming bugs; CPrint™ reformats, beautifies and documents C/C++ source files; and more... Works with MPW C/C++, THINK C, requires Apple's MPW. $295

CLImate by Orchard Software is a command line interface that lets you communicate with your Macintosh using English commands to create, delete, rename, and move files and folders. It can start applications, format disks, restart your computer, etc. CLImate supplements the Finder. It includes a BASIC interpreter that lets you script your Macintosh without AppleScript. The interpreter includes advanced programming constructs: repeat loops, if/then/else conditionals, subroutine calls, etc... CLImate implements wildcard characters, enabling you to work on groups of files. Use CLImate instead of MPW to manage your projects. CLImate is an application occupying 70K disk space. It comes bundled with sample programs and full documentation. $59.95

CMaster 2.0 by Jersey Scientific installs into THINK C 5 / 6 / 7 and Symantec C++ for Macintosh, and enhances the editor. Use its function popup to select a function and CMaster takes you right to it. Other features include multiple clipboards and markers, a Function Prototyper, and a GoBack Menu which can take you back to previous editing contexts. Almost all features bindable to the keyboard, along over a hundred keyboard-only features like "Add New Automatic Variable." Glossaries, AppleScript and ToolServer support, Macros, and External Tools you create too! $129.95

Cron Manager by Orchard Software implements the UNIX Cron facility. It can open any Macintosh file on a given date and time. By creating an alias, renaming it to the date and time to open, and moving it into the special Cron Events Folder, Cron Manager will open it. Cron Manager is a control panel that creates the special Cron Events Folder inside your System Folder. It is completely transparent to the user. It works like the Startup Items folder, only smarter. It works with any Macintosh file: if you can double-click to start it, Cron Manager can open it. $26.95. Cron Manager bundled with CLImate, $59.95

Dialog Maker by Electric Software Corporation. Migrating from C to C++? Dialog Maker can ease your transition. Dialog Maker is an object-oriented programming library for MPW C++ and Symantec C++ (MPW and Symantec Development Environment versions) which contains a complete set of routines that create a high level interface to dialogs. Dialog Maker provides a small number of simple, yet powerful routines to access

and manipulate dialogs. Resources are used to control the most common dialog behavior allowing you to develop your application lightning fast. Minimum requirements System 7.0, MPW 3.2, MPW C++ 3.2, or Symantec C++ 6.0. $149

InstallerPack™ by StepUp Software is a package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts. Compression formats supported are Compact Pro & Diamond. Each atom also available separately: $219

Last Resort Programmer's Edition records every keystroke, command key and mouse event (in local coordinates) to a file on your hard disk. This is especially useful for program testing & debugging, and for technical support and help desks. If something goes wrong (because of a power failure, system crash, forgetting to save or deleting lines) and you lose a word, phrase, or document you can look in the Last Resort keystroke file and recover what you typed. Last Resort is also useful for technical support personnel, when they have to ask "What was the last thing you did before...?" $74.95

LS Object Pascal CD includes the world's first Object Pascal compiler for Power Macintosh. 100% compatible with Apple's MPW Pascal, LS Object Pascal combines the best of Apple's native development tools with innovative new technology developed at Language Systems. Compiler options specify 68K or native PowerPC code generation. Included on the CD are: LS Object Pascal compiler, Universal Pascal Toolbox interfaces, fully loaded MPW 3.3.1, 68K and PowerPC source debuggers, PowerPC assembler, online documentation, Macintosh Tech Notes, and a special version of AppMaker by Bowers Development that generates native Pascal source code. The beta release includes upgrades to v1.0 when it becomes available. $399

Spellswell 7 1.0.4 is an award-winning, comprehensive, practical spelling checker that works in batch mode or within applications that incorporate the Apple Events Word Services protocol (e.g., Eudora, WordPerfect, Communicate!, and Fair Witness). Spellswell 7 checks for spelling errors as well as common typos like capitalization errors, spaces before punctuation, double double word errors, abbreviation errors, mixed case errors, extra spaces between words, a/an before vowel/consonant, etc... MacTech orders include developer kit with Writeswell Jr., a sample Apple Events Word Services word-processor and its source code. $74.95

MacAnalyst by Excel Software supports software engineering methods including structured analysis, data modeling, screen prototyping, object-oriented analysis, and data dictionary. This language independent tool is used by system analysts and software designers. Demo $79, Product $995

MacAnalyst/Expert by Excel Software supports software engineering methods with the capabilities of MacAnalyst plus state transition diagrams, state transition tables, decision tables and process activation tables. An integrated requirement database provides traceability from requirement statements to analysis or design diagrams, code or test procedures. This tool is well suited to the analysis and design of real-time or requirements driven projects. Demo $79, Product $1595

MacDesigner by Excel Software supports software engineering methods including structured design, object-oriented design, data dictionary and code browsing. This

Want more product info? E-mail us at productinfo@xplain.com

MAIL ORDER STORE

tool is well suited to detailed design or maintenance of software development projects. Demo $79, Product $995

MacDesigner/Expert by Excel Software supports software engineering methods with the capabilities of MacDesigner plus multi-task design. An integrated requirement database provides traceability from requirement statements to design diagrams, code or test procedures. This tool is well suited to design or maintenance of real-time, multi-tasking software projects. Demo $79, Product $1595

MacA&D by Excel Software combines the capabilities of MacAnalyst/Expert and MacDesigner/Expert into a single application. It supports structured analysis and design, object-oriented analysis and design, real-time extensions, task design, data modeling, screen prototyping, code editing and browsing, reengineering, requirement traceability, and a global data dictionary. Demo $149, Product $2995

MacWireFrame by Amplified Intelligence. Create your own virtual reality application with MacWireFrame, a virtual reality application frame work. Includes a complete library of object oriented graphics routines, its own easy to understand application frame work (similar to MacApp or TCL but a lot easier to understand), plus an example application program that lets you start solid modeling right away. Comes complete with fully documented source code. All new purchases will be guaranteed a $49.99 upgrade to the soon to be released, scriptable, MacWireFrame 5.0. Due to the overwhelming response the special price offer has been extended for a little while longer. **Special Offer: ~~$299.00~~ $75!!!!**

McCLint™ Rev. 2.2 by MMCAD. McCLint locates questionable C programming constructs, saving you hours by identifying programming mistakes and latent programming bugs. Some of the checks include variable type usage, conditional and assignment statement usage, arithmetic operations in conditional expressions, misplaced semicolons, pointer type coercion, function argument passing (with and without function prototypes), local and global variable initialization and usage, and existence/shape of return statements. McCLint includes a THINK C like, multiple window editor and source code highlighting system in a fully integrated environment. One or more files can be analyzed in an interactive or batch fashion. Works with THINK C (including OOPS), MPW C,... $149.95

McCPrint™ Rev 2.2 by MMCAD. McCPrint reformats and beautifies C and C++ source code in a user specified manner. You can transform code to and from your programming style, making source code easier to read and work with. In addition to code formatting, documentation support aids include source code pagination, line number inclusion and control flow graphing. McCPrint includes a multiple window editor and source code highlighting system in a fully integrated environment. Works with THINK C, MPW C/C++, supports System 7 and 32 bit addressing for use on any system including the Quadras. $99.95

The Memory Mine™ by Adianta is a stand alone debugging tool for Macintosh and native PowerPC. Programmers can monitor heaps, identify problems such as memory leaks, and stress test applications. Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is shown, as are heap corruption, fragmentation, and more... Allocate, Purge, Compact, and Zap memory let users stress test all or part of a program. Source code is not needed to view heaps. It works on Macintoshes with 68020 or later and System 7.0 or later. $99

p1 Modula-2 V5.1 is a full implementation of the ISO Standard for Modula-2 which includes exception handling, termination, complex numbers, value constructors, a standard library and more. In addition it supports objects and MacApp, foreign language calls, all current MPW interfaces, optimized 680x0 instructions, three floating point types with four modes of operation, etc. A symbolic window debugger, several utilities and a set of examples (including MacApp tutorial) are included. p1 Modula-2 requires MPW. It is targeted for professional development and prompt technical support by e-mail or FAX is granted. $395, corporate 5 pack $1175

PictureCDEF by Paradigm Software is a professional-level CDEF for creating custom "puffy" graphical buttons (8-64 pixels in size). PictureCDEF is multi-monitor and bit-depth sensitive. The button graphic (created with ResEdit) can be changed at runtime and even animated with a call-back routine. Create distinct buttons in six variations: PushButton, FlexiButton, ToggleButton, ChkButton, PushPictButton and TogglePictButton. Position the optional button title at left, bottom or right, or follow the system text direction for international support. 25 button frames, manual and sample code included. MacApp 3.0 support. Demo available on request. Full source code: $95. Object code only: $45.

Qd3d/3dPane/SmartPane source code bundle by Vivistar Consulting. **Qd3d 2.0:** Full featured 3d graphics. Points; lines; polygons; polyhedra; Gouraud shading; z-buffering; culling; depth cueing; parallel, perspective, and stereoscopic projections; performance enhancing "OnlyQD" and "Wireframe" modes; full clipping; pipeline access; animation and model interaction support; and a "triad mouse" to map 2d mouse movement to 3d. **3dPane 2.0:** Integrates Qd3d with the TCL and provides a view orientation controller. **SmartPane:** Offscreen image buffering, flicker free animation, and QuickTime movie recording. For use with Qd3d/3dPane or in 2d settings. All work with C++ compilers or ThinkC 6. $192

NEW! **QC™** by Onyx Technology, is a system extension that stress tests code during runtime for common and not-so-common errors. Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking. QC is extremely user friendly for the non-technical tester yet offers an API for programmers who want precise control over testing. $99

QUED/M 2.6 by Nisus is the text editor that has become the industry standard for speed and efficiency. This 32-bit clean version fully supports the MPW ToolServer through Apple Events, and the CODE module feature allows you to implement external source code as a menu command. In addition, QUED/M's macro facility lets you create customized menu commands. We've even improved our acclaimed Find and Replace feature which can search unopened files for literal text or regular expressions created with GREP metacharacters. New features include a file comparison option which combines two or three files into one and marks conflicts automatically. $149

ScriptGen Pro™ by StepUp Software is an Installer script generator which requires no programming or knowledge of Rez. Supports StepUp's InstallerPack, StuffIt compression, custom packages, splash screens, network installs, Rez code output, importing resources, and AppleEvent link w/MPW: $169

SoftPolish by Language Systems is a development tool that helps software developers avoid embarrassing spelling errors, detect incorrect or incompatible resources and improve the appearance of their Macintosh software. SoftPolish examines application resources and reports potential problems to a scrolling log. Independent of any programming language or environment, SoftPolish improves the quality of any Macintosh program. $169

NEW! **Spyer** by InCider is a simple operated tool that records all actions (including mouse movement) you perform on a Macintosh computer and then replays them at your preferred speed. The recorded data can be saved in files for future use. Spyer works as a background process with any Macintosh application and is triggered by user defined Hot Keys. Spyer enables the "Continuous Redo" utility and is especially useful for software testing and demonstration. $39

StoneTable: A library replacing all functions found in list manager plus: variable size columns/rows; different font, size, style, forecolor, backcolor per cell; sort, resize, move, copy, hide columns/rows; edit cells/titles in place; titles for columns/rows; multiple lines per cell; grid line pattern/color; greater than 32k data per table; up to 32k text per cell; support for balloon help and binary cell data. Versions for Think C, Think Pascal, MPW C, MPW Pascal, CodeWarrior C. (all prices per developer) $150 first compiler, additional compilers $50

Stone Table Extra: Additional functions for StoneTable. Drag selected cells within table or to other tables; optionally add rows as part of drag; popup menus or check boxes in cells; variable width grid lines; move/drag/resize table in window; clipboard operations on multiple cells. Requires StoneTable. (all prices per developer) $50 first compiler, additional compilers $25

NEW! **StoneTable and StoneTableExtra for PowerPC:** Same functionality as 68K libraries. Versions for MPW C and CodeWarrior C. Must have 68K libraries. (all prices per developer) StoneTable $100, StoneTableExtra $25

ViperBase by Viper Development is a fast database designed for developers that want speed but don't want to spend months or years developing a commercial quality database. ViperBase: Unlimited Records, Variable Length: $59. ViperBase II: ViperBase + Multiple Indices. $119

DON'T FORGET!

To receive information on any products advertised in this issue, send your request via Internet: productinfo@xplain.com

Want more product info? Call us at 310/575-4343.

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

Absoft	51
Accusoft	15
ACI US, Inc.	33
Adianta Inc.	24
Aladdin Knowledge Systems Ltd.	7
Apple Developer University	47
AppleSoft Sales Group	13
Ariel Publishing	83
BareBones Software	43
Bateman, Inc.	87
Celestin Company	19
Creative Solutions	28
DataPak Software, Inc.	39
Douglas Electronics, Inc.	16
dtF Americas, Inc.	36
Foundation Solutions	34
Graphic Magic	68
Graphical Business Interfaces Inc.	26 & 27
InCider International	83
Jasik Designs	17
Jersey Scientific	35
Language Systems	34
Lextek International	67
MacHack Conference	68
MacTech CD-ROM, Vol 8	88
MacTech Mail Order Store	49
MacWorld Exposition	40
MacXperts	87
Mainstay	1
Metrowerks	8
MindVision Software	20
Neologic Systems	25
Nisus	50
Onyx Technology	66
OPtions Computer Consulting	30
PACE Anti-Piracy	45
POET Software	5
Quasar Knowledge Systems	IBC
Rainbow Technologies	11
Ray Sauers Associates	71
Richey Software Training	67
Scientific Placement	87
Sierra Software Innovations	BC
SNA, Inc.	46
StepUp Software	23
Stone Tablet Publishing	59
Symantec	IFC
TSE International	29
Vermont Database Corporation	31
Water's Edge Software	22

LIST OF PRODUCTS

Accusoft Image Format Library • Accusoft	15
Apprentice • Celestin Company	19
BBEdit 3.0 • BareBones Software	43
4D TOOLKIT 2.0 • OPtions Computer Consulting	30
Cataloger™ • Graphical Business Interfaces Inc.	26 & 27
CMaster® • Jersey Scientific	35
CodeWarrior™ • Metrowerks	8
CXBase Pro • TSE International	29
Database Scripting Kit™ • Graphical Business Interfaces Inc.	26 & 27
The Debugger V2 • Jasik Designs	17
Developer Vise 3.0 • MindVision Software	20
DragInstall 1.5.3 • Ray Sauers Associates	71
EHelp 4.0 • Foundation Solutions	34
Fortran 77 Development Tool • Absoft	51
IcePick 3.0™ • Sierra Software Innovations	BC
Inside Out II® • Sierra Software Innovations	BC
Installer Pack • StepUp Software	23
LS Object Pascal/PPC • Language Systems	34
MacApp Programmers • MacXperts	87
Mac Disk Duplicator • Douglas Electronics, Inc.	16
MacForth Plus 4.2 • Creative Solutions	28
MacHack™ Conference	70
MacHASP® • Aladdin Knowledge Systems Ltd.	7
MacEncrypt™ • PACE Anti-Piracy	45
MacNosy • Jasik Designs	17
MacRegistry™ • Scientific Placement	87
MacWorld Expo • MacWorld Exposition	40
neoAccess™ • Neologic Systems	25
Object Master • ACI US, Inc.	33
OP2CPLUS • Graphic Magic	68
p2cII™ • Sierra Software Innovations	BC
PAIGE™ • DataPak Software, Inc.	39
PatchWorks™ • SNA, Inc.	46
Pinnacle Relational Engine • Vermont Database Corporation	31
POET Object Database for C++ • POET Software	5
Power PC Training • Apple Developer University	47
Programmer Training • Richey Software Training	67
QC: The Macintosh Testing Solution • Onyx Technology	66
QDFX™ • Ariel Publishing	83
QUED/M 2.7 • Nisus	50
Recruitment • Bateman, Inc.	87
Recruitment • MacXperts	87
Recruitment • Scientific Placement	87
Relational Database System • dtF Americas, Inc.	36
SALESBASE 2.1™ • Sierra Software Innovations	BC
ScriptGen Pro • StepUp Software	23
Sentinel™ • Rainbow Technologies	11
Sierra Consulting Group • Sierra Software Innovations	BC
SmalltalkAgents® • Quasar Knowledge Systems	IBC
Spellchecker Toolkit • Lextek International	67
Spyer • InCider International	83
Stone Table™ • Stone Tablet Publishing	59
Symantec C++ 7.0 • Symantec	IFC
Template Constructor™ • Graphical Business Interfaces Inc.	26 & 27
The Memory Mine™ • Adianta Inc.	24
Tool Plus™ 2.5 • Water's Edge Software	22
VIP-C • Mainstay	1

TIPS & TIDBITS

By Scott T Boyd, Editor

TIP OF THE MONTH

WHITE-BORDERED BLACK TEXT MODE

A recent thread on comp.sys.mac.programmer prompted this little code snippet to work around a glaring omission on QuickDraw text modes. This is actual, tested, working code to draw a white outline around black characters. The trick is in the use of CharExtra, a Color QuickDraw routine that's worth reading about in *Inside Macintosh*.

```
void
Outline (
        short x ,
        short y ,
        Ptr text ,
        short length )
{
        long wl , w2 , ce ;

        TextMode ( srcBic ) ;
        TextFace ( outline ) ;
        wl = TextWidth ( text , 0 , length ) ;
        MoveTo ( x , y ) ;
        DrawText ( text , 0 , length ) ;

        TextMode ( srcOr ) ;
        TextFace ( normal ) ;
        w2 = TextWidth ( text , 0 , length ) ;
        ce = ( wl - w2 ) * 0x10000L / length ;
        CharExtra ( ce ) ;
        SpaceExtra ( ce ) ;
        MoveTo ( x , y ) ;
        DrawText ( text , 0 , length ) ;

        CharExtra ( 0 ) ;
        SpaceExtra ( 0 ) ;
}
```

It only works in color GrafPorts, but that's to be expected since CharExtra requires Color Quickdraw.

— Jon Wätte (h+@nada.kth.se)
Hagagatan 1, 113 48 Stockholm, Sweden

Send us your tips! We'll send you money, and developers all over the world (even in Sweden) will marvel at your insight, your wisdom, or the simple fact that you've got enough extra time to write and send a little bit of e-mail to make their lives a little bit better.

We pay $25 for every tip we use, and $50 for Tip of the Month. You can take your award in orders or subscriptions if you prefer. Make sure code compiles, and send tips by e-mail; editorial@xplain.com is a great address. See page two for our other addresses.

AUNTIE FLICKER

Many operations, especially moving and sizing of controls, cause a lot of flicker on the screen. Some of this can be avoided by hiding the control, moving and sizing it, and bringing it back, but even then there is more drawing to the screen than is really necessary.

The trick to avoiding all this unnecessary drawing is to temporarily turn off drawing during the operations which cause flicker, and then turn it on again. To do this, simply replace the visRgn with a null region temporarily, i.e.:

continued to the right…

```
RgnHandle vis;

...

vis = window->visRgn;
window->visRgn = NewRgn();
```

... operations which would normally cause flicker ...

```
DisposeRgn(window->visRgn);
window->visRgn = vis;
```

... don't forget to redraw or invalidate appropriately here!

— Jorg 'jbx' Brown
San Francisco, CA

The Future is Dynamic, Is Your Development Environment?

- ♦ Dynamic environment with incremental compiling. NO compile-link downtime.
- ♦ Intelligent, integrated UI Components. NO dumb resources separate from your code.
- ♦ Integrated Object-Linker provides component support today, NOT tomorrow.
- ♦ High-level support for System 7.x makes it easy to leverage Apple technology.
- ♦ Simplified access to databases via DAL/DAM. Sample code included.
- ♦ Inline foreign function callbacks to C, Pascal, Fortran and Assembly.
- ♦ Fully styled source code with fonts and color. WorldScript™ support included.
- ♦ HyperCard™ XMCD interface to assist HyperCard developers "moving up."
- ♦ Cross-platform development with full support of native OS functionality.
- ♦ Scalable development, from simple utilties to complex distributed applications.

SmalltalkAgents® is *the* Interactive OODL

SmalltalkAgents® is the *premier* commercial object-oriented dynamic development environment. New key features of SmalltalkAgents Professional include:

NEW Documentation
Features the Macintosh® version of Dan Shafer's *Smalltalk Programming for Windows* book, specifically re-written for SmalltalkAgents, and four other complete volumes including "Road Map", "A Quick Trip to ObjectLand", "Developer Guide: Workbench", and "Developer Guide: Class Frameworks".

NEW GUI Builder
Live "Drag and Drop" manipulation to build your application's visual interface using smart components that "know" how to behave. Wire together components and save designs as reusable templates that can be dropped into other applications or components.

NEW External Code Linking Toolkit
Preserve your investment in existing C, Pascal, and Fortran code by dynamically linking your routines into the SmalltalkAgents integrated development environment, including optional object "wrappers".

SmalltalkAgents Lite now available for $95 direct from QKS!

SmalltalkAgents
The Power to Create a New Generation of Software for a New Generation of Macintosh
PRODUCTIVITY ENHANCED
QKS

Buy SmalltalkAgents Pro now for discount on native Power Macintosh version!*
(*Shipping January 1995, offer ends 12/31/94)

The Next Generation Tool for Cross-Platform Object-Oriented Programming, including a superset of Smalltalk... the language of choice for corporate development!

1-800-296-1339
or 301-530-4853 (info@qks.com)

QKS

Quasar Knowledge Systems • 9818 Parkwood Drive • Bethesda, Maryland 20814-4028 USA • Phone: 301-530-4853 • Fax: 301-530-5712

Sierra Software Innovations

Six Years of Quality Products and Services

SALESBASE 2.1™
Fully Integrated Sales Automation Solution for Large Corporations

Inside Out II®
Industrial Strength Multi-User Relational Database Engine

p2cII™
Advanced Object Pascal to C++ Source Code Translator

IcePick 3.0™
Original Award Winning MacApp 2.x and 3.x View Editor

Sierra Consulting Group
Top-Notch DataBase and SalesAutomation Consulting

Please call 800.621.0631 for a complete information package on Sierra's line of prestigious products and custom services.

Better Software through Innovation

SIERRA SOFTWARE INNOVATIONS

923 Tahoe Blvd., Incline Village, Nevada 89451
Phone: 702.832.0300 Fax: 702.832.7753 Internet: D2086@applelink.apple.com